Globalisation de méthodes d'ordre supérieur en optimisation non linéaire sans contraintes

par

Samuel GOYETTE

Mémoire présenté au Département de mathématiques
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, janvier 2020

Le 8 janvier 2020

le jury a accepté le mémoire de Monsieur Samuel Goyette dans sa version finale.

Membres du jury

Professeur Jean-Pierre Dussault
Directeur de recherche
Département d'informatique

Professeur Dominique Orban
Codirecteur de recherche
Département de mathématiques, Polytechnique Montréal

Professeur François Dubeau
Membre interne
Département de mathématique

Professeur Virginie Charette
Président-rapporteur
Département de mathématiques

SOMMAIRE

Ce mémoire s'attarde sur la résolution du problème d'optimisation non-linéaire et sans contrainte. Les éléments de base de l'optimisation non linéaire sont introduits. Les méthodes de Newton, Chebyshev, Halley et Shamanskii sont présentées, ainsi que les détails d'implémentation numérique relatives à ces méthodes. Les algorithmes de région de confiance et de régularisation cubique sont également présentés et de nouvelles idées pour introduire les directions de Chebyshev, Halley et Shamanskii dans ces algorithmes sont mises en évidence. De nombreux exemples numériques accompagnent les preuves théoriques pour assurer un pont entre la théorie et la pratique.

REMERCIEMENTS

Ce mémoire est non seulement l'aboutissement de deux ans de maîtrise, mais aussi de mon parcours universitaire. J'aimerais pouvoir remercier tous ceux qui m'ont aidé pendant les dernières années.

Tout d'abord, il va sans dire que sans l'appui de mes superviseurs, Jean-Pierre Dussault et Dominique Orban, cette maîtrise n'aurait jamais convergé. Grâce à vos conseils, vos idées et votre support, je suis devenu un meilleur mathématicien et un meilleur scientifique. Merci de m'avoir permis de vivre une expérience de recherche aussi passionnante.

Je remercie mes collègues du BISOUS qui m'ont offert un des meilleurs lieux de travail possible. Pendant mes stages de recherche ou ma maîtrise, votre support moral et bisouslosophique m'a grandement aidé dans les dernières années. J'aimerais donc remercier Adam, Émilie, Jonathan, Luc, Mathieu, Maxime, Paul et Tangi. Je porterai toujours fièrement le titre de S.A.M. du BISOUS.

Je remercie Alexis ainsi que tous ceux qui m'ont aidé lors de mon passage au GERAD. J'ai passé une très belle session à Montréal.

Étant donné qu'il m'est arrivé de sortir du laboratoire de temps à autre, j'aimerais remercier les personnes suivantes qui ont été là pour moi dans les bons et moins bons moments depuis le début de mes études universitaires : Catherine, Denis, Jean-Christophe, Ga-

brielle, Guillaume et Manpreet.

Mes parents, Jean Goyette et Marie Champagne, m'ont toujours supporté et encouragé lors de mes études. Leur support moral, financier, émotionnel ou culinaire (voir gastronomique) a été inconditionnel. Merci d'avoir toujours été là.

Finalement, je tiens à remercier Ting Wei Chuang. Il me tarde de commencer la prochaine étape de ma vie à tes côtés.

Samuel Goyette

Sherbrooke, Novembre 2019

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION

On désire résoudre le problème d'optimisation non linéaire sans contraintes

$$\min_{x \in \mathbb{R}^n} f(x) \tag{1}$$

où $f : \mathbb{R}^n \to \mathbb{R}$ et $f \in \mathcal{C}^2$ ou $\mathcal{C}^3$. Ce problème peut être la modélisation d'une multitude de situations, que ce soit en apprentissage automatique, en imagerie médicale ou encore une version modifiée d'un problème d'optimisation avec contraintes. Le calcul différentiel introduit au 17e siècle par Newton et Leibniz introduit les idées de base pour résoudre ce problème de manière directe dans le cas à une seule variable [SS40, New69, Rap97]. En effet, on dérive la fonction objectif et regarde les racines de sa dérivée et le signe de la dérivée seconde. Or, cette façon de faire est très limitée. Si f n'a qu'une seule variable, il n'existe pas toujours de formule pour déterminer les racines de la dérivée. Ce problème devient encore plus complexe lorsque $f : \mathbb{R}^n \to \mathbb{R}$. On doit donc développer des méthodes numériques qui approximent une solution.

Newton et Raphson ont développé les premières méthodes pour résoudre ce problème de manière itérative. Leurs méthodes utilisent les première et deuxième dérivées de f et offre une convergence quadratique vers la solution. Halley s'est intéressé a ce problème à la fin du 16e siècle [Hal94] et propose une méthode qui enrichit la méthode de Newton. Au milieu du 19e siècle, Chebyshev produit une méthode similaire à celle de Halley [CMS99]. Une autre méthode de résolution de ce problème s'inspirant de la méthode de Newton

a été proposée au milieu du 20e siècle par Shamanskii [Š67]. Les méthodes de Halley, Chebyshev et Shamanskii sont des variantes qui apportent toutes une correction à la méthode de Newton traditionnelle. Les méthodes de Halley et Chebyshev utilisent des dérivées d'ordre trois pour obtenir cette correction, tandis que la méthode de Shamanskii se limite à des dérivées d'ordre deux. Le point commun de ces méthodes est qu'elles offrent une convergence cubique vers la solution. On définit les méthodes d'ordre supérieur comme des méthodes qui offrent une convergence cubique ou supérieure vers la solution. La grande difficulté liée à certaines de ces méthodes est le calcul de dérivées d'ordre trois. Ces différents algorithmes garantissent d'atteindre une solution tant qu'une estimation initiale de la solution dans un certain voisinage de la solution optimale est fournie. Il s'agit de la convergence locale.

La principale difficulté calculatoire de la méthode de Newton est le calcul de la hessienne (la matrice des dérivées secondes). Il existe une classe de méthode, les méthodes quasi-Newton, qui approximent la hessienne ou l'inverse de la hessienne. Ces méthodes n'évaluent donc jamais de dérivées secondes, mais utilisent seulement de l'information issue des premières dérivées. Parmi les méthodes de quasi-Newton, on retrouve l'algorithme de Broyden-Fletcher-Goldfarb-Shanno (BFGS) introduit à la fin des années 60 [Bro70, Fle70, Gol70, Sha70]. Cette méthode construit une matrice H telle que son inverse est une approximation de la hessienne. Il existe également l'algorithme L-BFGS qui est une variante de BFGS, le « L » signifie *limited memory* (mémoire limitée). L'algorithme Davidon-Fletcher-Powell (DFP) est un algorithme de quasi-Newton similaire à la méthode BFGS.

On peut étudier (1) d'un point de vue de la convergence globale, c'est-à-dire la recherche d'une solution à partir d'un estimé initial quelconque. Une des stratégies de globalisation est de considérer des méthodes de type région de confiance. Conn, Gould et Toint [CGT00] retracent et présentent les méthodes de région de confiance. En 2011, Cartis, Gould et

Toint proposent les méthodes adaptatives par régularisation cubique (ARC) [CGT11a, CGT11b] qui partagent certaines similarités avec les méthodes de région de confiance.

Une autre stratégie de globalisation est la recherche linéaire. Cette stratégie consiste à regarder le comportement de la fonction objectif dans une restriction linéaire donnée par une direction de descente. Armijo [Arm66] et Wolfe [Wol69] font partie des pionniers de cette stratégie. La recherche linéaire est la manière principale de globaliser les méthodes de quasi-Newton. Traditionnellement, les techniques de globalisation visent à se ramener le plus rapidement possible à la méthode de Newton (ou ses approximations). Les méthodes d'ordre supérieur ne sont presque jamais utilisées dans des schémas de globalisation. La stratégie de Lampariello et Sciandrone [LS01] constitue la manière principale de globaliser la méthode de Shamanskii.

Dans ce mémoire, nous proposons de nouvelles idées pour introduire les méthodes d'ordre supérieur dans les schémas connus de globalisation de la méthode de Newton.

Dans le premier chapitre, nous présenterons quelques préliminaires de l'optimisation non linéaire et introduirons les notations utilisées dans la suite de ce mémoire. Dans le deuxième chapitre, nous étudierons les propriétés de convergence locale des méthodes de Newton, Halley, Chebyshev et Shamanskii en détail ainsi que les différentes problématiques liées à leurs implémentations. Dans le troisième chapitre, nous verrons les différentes techniques de globalisation en nous attardant aux algorithmes de région de confiance et ARC. Nous regarderons les preuves de convergence de ces différentes méthodes afin de voir comment y introduire les directions d'ordre supérieur. Le quatrième chapitre constitue le cœur des recherches effectuées. Nous y verrons différentes stratégies de globalisation de méthodes d'ordre supérieur. Nous présenterons différents résultats numériques pour assurer le pont entre la théorie et la pratique.

CHAPITRE 1

Mise en contexte et préliminaires

Dans cette section, nous rappellerons quelques éléments de base de l'optimisation non linéaire. Nous nous contenterons de rappeler les concepts nécessaires à la résolution de (1). Nous analyserons les conditions d'optimalité que doivent satisfaire les solutions de (1). Nous rappellerons les différents éléments d'analyse nécessaires en optimisation non linéaire ainsi que les outils d'algèbre linéaire numérique nécessaires par la suite. Finalement, nous étudierons différentes stratégies pour calculer les dérivées de fonctions à plusieurs variables.

Pour l'ensemble de ce document, I représente la matrice identité $n \times n$. À moins d'indication contraire, les vecteurs sont des vecteurs colonnes.

1.1 Conditions d'optimalité

Afin de résoudre (1), commençons par définir une solution de ce problème.

Définition 1.1 *Un minimum local x^* de f est tel que pour tout x dans un voisinage*

(une boule ouverte) autour de x^* *de rayon* ϵ *on a* $f(x^*) \leqslant f(x+\epsilon)$.

Il est également possible de définir un minimum global de f.

Définition 1.2 *Un minimum global* x^* *de* f *est tel que pour tout* x *dans* $\mathbb{R}^n$ *on a* $f(x^*) \leqslant f(x)$.

Dans ce document, une solution fait référence à un minimum local. Les algorithmes introduits dans les chapitres 2 et 3 visent à atteindre un minimum local, cependant ces algorithmes garantissent seulement de trouver un point stationnaire. La recherche de minima globaux n'est pas abordée.

Définition 1.3 *Un point* x^* *qui annule le gradient de* f*, i.e* $\nabla f(x^*) = 0$*, est un point stationnaire de* f.

Les conditions d'optimalité de (1) sont un ensemble d'équations et d'inéquations qui indiquent les propriétés qu'une solution x^* doit respecter. Les conditions nécessaires d'optimalité sont présentées dans le théorème 1.4.

Théorème 1.4 *Soit* $f \in \mathcal{C}^2$ *et* x^* *une solution de (1), alors* x^* *est un point stationnaire de* f *et la hessienne est semi-définie positive en* x^* *(*$\nabla^2 f(x^*) \succeq 0$*).*

On note que $\nabla f(x^*) = 0$ est la condition nécessaire de premier ordre et $\nabla^2 f(x^*) \succeq 0$ est la condition nécessaire de second ordre. Les conditions suffisantes d'optimalité sont présentées dans le théorème 1.5.

Théorème 1.5 *Soit* x^* *tel que* $\nabla f(x^*) = 0$ *et la hessienne est définie positive en* x^* *(*$\nabla^2 f(x^*) \succ 0$*), alors* x^* *est une solution de (1).*

Les preuves pour les théorèmes 1.4 et 1.5 sont disponibles dans [Fle87].

1.2 Analyse numérique

Un des outils les plus importants pour l'optimisation non linéaire est le développement de Taylor, en particulier le développement limité d'ordre deux :

$$q_x(d) = f(x) + \nabla f(x)^T d + \tfrac{1}{2} d^T \nabla^2 f(x) d. \tag{1.1}$$

Ce développement représente une approximation quadratique de f autour de x. Notons que

$$\nabla q_x(d) = \nabla f(x) + \nabla^2 f(x) d.$$

Donc,

$$\nabla q_x(d) = 0 \iff \nabla f(x) + \nabla^2 f(x) d = 0, \tag{1.2}$$

$$\iff \nabla^2 f(x) d = -\nabla f(x). \tag{1.3}$$

Ainsi si $\nabla^2 f(x) \succ 0$ le minimum de $q_x(d)$ est atteint en d tel que $\nabla^2 f(x) d = -\nabla f(x)$.

Les notations de Landau sont un autre outil important, il s'agit d'un ensemble de notations qui permettent de comparer différentes fonctions entre elles. Nous nous attardons seulement sur les notations o et $\mathcal{O}$. Nous prenons les définitions directement de [Dus15]. Ces notations nous permettront de comparer la vitesse de convergence des algorithmes étudiés dans le reste du mémoire.

Définition 1.6 *Soient deux fonctions f et g telles que $f(\bar{x}) = g(\bar{x}) = 0$. Nous notons $f \sim \mathcal{O}(g)$ (f est grand O de g) si et seulement si il existe $\gamma \geqslant 0$ tel que*

$$\limsup_{x \to \bar{x}} \left| \frac{f(x)}{g(x)} \right| = \gamma < \infty.$$

De manière alternative, si $\{f(x_n)\}_{n\geqslant 0}$ et $\{g(x_n)\}_{n\geqslant 0}$ sont des suites on peut dire que $f \sim \mathcal{O}(g)$ s'il existe $k \in (0,1)$, n_0 tel que pour tout $n \geqslant n_0$ on a $|f(x_n)| \leqslant k\,|g(x_n)|$. La notation o est très similaire.

Définition 1.7 *Nous notons $f \sim o(g)$ (f est petit o de g) si et seulement si :*

$$\limsup_{x\to\bar{x}} \left|\frac{f(x)}{g(x)}\right| = 0$$

On note que si $f \sim o(g)$ alors $f \sim \mathcal{O}(g)$. De plus, si $f \sim \mathcal{O}(g)$ et $g \sim \mathcal{O}(f)$ alors on utilise la notation $f \sim g$.

1.3 Méthodes numériques en algèbre linéaire

Résoudre un système de la forme $Ax = b$ où A est $n \times n$ est une étape cruciale de tous les algorithmes étudiés dans ce travail. En théorie, la résolution de systèmes linéaires est connue depuis longtemps. Cependant, certaines considérations numériques rendent incommode l'inversion de A. Inverser une matrice est une opération d'ordre $\mathcal{O}(n^3)$ en termes d'opérations élémentaires (+, -, * et /). Pour inverser une matrice A en $\mathcal{O}(n^3)$ opérations il faut décomposer la matrice A et résoudre n systèmes linéaires de la forme $Ax_i = e_i$ où e_i est la i-ème colonne de la matrice identité. Ainsi, il est possible de construire A^{-1} colonne par colonne. Cette façon de faire est contre-productive. En effet, si A est décomposée de telle sorte que le système $Ax_i = e_i$ est facile à résoudre, il est plus efficace de simplement résoudre $Ax = b$ directement. Nous qualifions un système linéaire de « facile » à résoudre si A est :

- diagonale, i.e. les seuls éléments non nuls de A sont sur sa diagonale ;
- triangulaire, i.e. les seuls éléments non nuls de A sont au dessus ou en dessous de la diagonale ;

— orthogonale, i.e. A est telle que $A^TA = AA^T = I$.

De nombreuses méthodes d'algèbre linéaire numérique ont été développées pour résoudre de manière efficace et peu coûteuse un système linéaire $Ax = b$. Nous en présentons deux : la méthode par décomposition LDL^T et la décomposition de Bunch-Kaufman. Nous supposons que A est une matrice symétrique.

Un autre aspect de la matrice A qui nous intéresse est sa proportion d'éléments nuls. Une matrice avec une faible proportion d'éléments non nuls est une matrice creuse tandis qu'une matrice avec peu d'éléments nuls est dite dense. Les matrices creuses sont généralement codées de manière différente des matrices denses pour sauver des allocations de mémoire inutiles. Le choix d'une représentation dense ou creuse pour la matrice A a donc une influence sur l'efficacité de la résolution du système linéaire.

1.3.1 Décomposition LDL^T

Les méthodes de solution directe consistent à décomposer la matrice A en facteurs. Ces sous-matrices sont utilisées pour résoudre des systèmes linéaires intermédiaires plus simples. Une fois que la matrice A est factorisée, il est possible de réutiliser la factorisation pour de nouveaux membres de droite. L'idée de réutiliser une factorisation est appliquée dans le calcul des directions d'ordre supérieur étudiées dans les sections 2.3 et 2.5.

Soit A symétrique et définie positive. On décompose A sous la forme LDL^T où L est une matrice triangulaire inférieure avec des 1 sur la diagonale et D est une matrice diagonale telle que les éléments diagonaux sont strictement positifs. Le système linéaire devient $LDL^Tx = b$, on pose $y = DL^Tx$. On a donc le système $Ly = b$. Il s'agit d'un système facile à résoudre. Par la suite, pour trouver x, on résout $Dz = y$ où $z = L^Tx$ et on résout un système triangulaire $L^Tx = z$.

La décomposition LDL^T est une variante de la décomposition de Cholesky qui décompose $A = LL^T$, où L est triangulaire inférieure.

Pour construire les matrices L et D, on utilise l'algorithme 1.

Algorithme 1 : Algorithme de décomposition LDL^T

```
1 pour i ← 1 à n faire
2     D_ii = A_ii − Σ_{j=1}^{i−1} L_ij² D_jj ;
3     pour j ← i + 1 à n faire
4         L_ji = ( A_ji − Σ_{k=1}^{i−1} L_jk L_ik D_kk ) / D_ii ;
```

Il existe différentes décompositions LDL^T qui répondent à différents besoins. L'algorithme 1 n'utilise pas de pivotage. Certaines variantes peuvent permuter des lignes et/ou colonnes pour s'assurer qu'on ne génère pas un D_{ii} nul. Le pivotage permet également de s'assurer que L est creuse si A est creuse. Il est donc préférable d'utiliser le pivotage lorsque l'on a affaire à des matrices creuses.

La décomposition de Cholesky et la décomposition LDL^T ont une complexité algorithmique de $\mathcal{O}(n^3)$ pour les matrices denses.

1.3.2 Décomposition de Bunch-Kaufman

Une des variantes de la décomposition LDL^T est celle de Bunch-Kaufman [BK77]. La matrice A est décomposée en un produit de trois matrices LDL^T où L est une matrice triangulaire inférieure, mais D est une matrice diagonale par blocs, c'est-à-dire que la diagonale de D est constitué de blocs 1×1 (un scalaire) ou 2×2. La décomposition

de Bunch-Kaufman produit également une matrice P de permutations et une matrice S diagonale de mise à l'échelle qui sont telles que $LDL^T = PSASP^T$.

Il existe également une version bornée de la décomposition de Bunch-Kaufman, notée BBK (*bounded Bunch-Kaufman*) [CH98]. Lorsque la version bornée est utilisée, il existe $0 < l < u < \infty$ tel que $l \leqslant \|LL^T\|_2 \leqslant u$. L'intérêt d'avoir de telles bornes est de limiter les effets négatifs causés par l'accumulation d'erreurs numériques. La variante de Bunch-Kaufman, tout comme la décomposition LDL^T, a une complexité algorithmique de $\mathcal{O}(n^3)$ pour les matrices denses.

1.4 Calcul de dérivées

Il existe différentes manières de calculer le gradient et la hessienne de f. Chaque méthode possède différentes propriétés, entre autres :

- la précision des dérivées (certaines méthodes calculent des dérivées exactes et d'autres ne calculent que des approximations) ;
- la facilité d'implémentation (certaines méthodes sont complexes à coder de manière efficace et robuste, par exemple la différentiation automatique. Cela limite souvent leur usage pour des dérivées).

Nous présentons 3 manières de calculer les dérivées qui offrent différents degrés de précision et différents niveaux de difficulté en termes d'implémentation.

1.4.1 Différences finies

En une dimension, la dérivée de f en x est définie par

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

Donc :

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \approx \frac{f(x+h) - f(x)}{h}.$$

En analyse numérique, cette façon d'approcher les dérivées est la méthode des différences finies progressives. On peut utiliser cette idée pour approcher des dérivées partielles de $f : \mathbb{R}^n \to \mathbb{R}$ et donc former le gradient de f. Il existe différentes variantes de ces méthodes (différences finies en régressives, progressives ou centrées). Des variantes pour les dérivées d'ordre 2 et supérieures existent. Cependant, ces méthodes ont toutes des inconvénients majeurs :

1. En théorie, elles ne produisent que des approximations de la dérivée. Il y a toujours un terme d'erreur, donc la dérivée produite n'est jamais exacte.

2. En pratique, même si un h très petit est utilisé (par exemple (10^{-15})) il y a des risques d'erreur de calcul numérique, entre autres, la soustraction de nombres « voisins ».

En guise d'exemple, si on prend $f(x) = e^x$, on peut voir la valeur de la dérivée calculée avec les différences finies progressives dans le tableau 1.1. La dérivée de e^x en $x = 1$ est connue ; il s'agit de e. Donc, on peut regarder l'erreur absolue de notre estimation par rapport à la véritable valeur de la dérivée.

On remarque que lorsqu'on prend h très petit, on perd de la précision. Le h optimal est $\mathcal{O}(\sqrt{\text{eps}(T)})$ où T est la façon dont les nombres sont encodés (demi-précision, simple précision, double précision). En double précision (`Float64`) le h optimal correspond à 10^{-8}. Donc, la méthode des différences finies est limitée pour calculer les gradients et les hessiennes de nos fonctions.

h	$f'(x)$	$\lvert f'(x) - e \rvert$
10^{-1}	2.858841954873883	10^{-1}
10^{-2}	2.7319186557871245	10^{-2}
10^{-3}	2.7196414225332255	10^{-3}
10^{-4}	2.718417747082924	10^{-4}
10^{-5}	2.7182954199567173	10^{-5}
10^{-6}	2.718283187430614	10^{-6}
10^{-7}	2.718281968405733	10^{-7}
10^{-8}	2.718281821856294	10^{-9}
10^{-9}	2.7182820439008983	10^{-7}
10^{-10}	2.71828337616852	10^{-6}
10^{-11}	2.7183144624132174	10^{-5}
10^{-12}	2.718714142702083	10^{-4}
10^{-13}	2.7178259642823828	10^{-3}
10^{-14}	2.7089441800853815	10^{-1}
10^{-15}	3.108624468950438	

Tableau 1.1 – Valeurs de $f'(x)$ calculées par différences finies

1.4.2 Nombre hyperduaux

Malgré le fait que les dérivées que l'on veut calculer soient réelles, il existe différentes méthodes utilisant d'autres ensembles de nombres pour calculer les dérivées. Par exemple, il existe la méthode du pas complexe [MSA03]. Soit $x, h \in \mathbb{R}$, i le nombre imaginaire (i.e. $i^2 = -1$) et ih un « pas unitaire complexe ». Alors, la série de Taylor de f autour de x est :

$$f(x+ih) = f(x) + ihf'(x) - h^2\frac{f''(x)}{2!} - ih^3\frac{f'''(x)}{3!} + \cdots . \tag{1.4}$$

En prenant la partie imaginaire de $f(x+ih)$ (on utilise la notation $\mathrm{Im}\,[f(x+ih)]$) il est possible d'isoler $f'(x)$. Donc,

$$\begin{aligned} f'(x) &= \frac{\mathrm{Im}\,[f(x+ih)]}{h} + h^2\frac{f'''(x)}{3!} + \cdots \\ &= \frac{\mathrm{Im}\,[f(x+ih)]}{h} + \mathcal{O}(h^2). \end{aligned}$$

Les valeurs de $f'(x)$ où $f(x) = e^x$ avec différentes valeurs de h sont présentées dans le tableau 1.2. On peut observer que plus h est petit, plus la valeur de la dérivée est précise. En effet, il n'y pas de soustraction de nombres voisins. Donc, la précision machine est atteinte pour une valeure de h suffisament petite.

Cependant, la méthode du pas complexe a des limitations similaires aux différences finies. En effet, il est possible d'obtenir des dérivées d'ordre un exactes avec h petit, mais pour les dérivées d'ordre deux, la méthode du pas complexe présente les mêmes difficultés que les différences finies, c'est-à-dire il ne s'agit que d'approximations numériques et il y a des erreurs numériques dues à la soustraction de nombres voisins.

Les nombres hyperduaux sont un ensemble de nombres pouvant être utilisés pour calculer des dérivées d'ordre un et deux de manière exacte [FA11]. Les nombres hyperduaux peuvent être vus comme une extension des nombres complexes. Les nombres complexes

h	$f'(x)$	$\lvert f'(x) - e \rvert$
10^{-1}	2.7137536234405366	10^{-1}
10^{-2}	2.718236523988427	10^{-5}
10^{-3}	2.7182813754120962	10^{-7}
10^{-4}	2.7182818239285753	10^{-9}
10^{-5}	2.7182818284137404	10^{-11}
10^{-6}	2.718281828458592	10^{-13}
10^{-7}	2.7182818284590406	10^{-15}
10^{-8}	2.718281828459045	0.0
10^{-9}	2.718281828459045	0.0
10^{-10}	2.718281828459045	0.0
10^{-11}	2.718281828459045	0.0
10^{-12}	2.718281828459045	0.0
10^{-13}	2.718281828459045	0.0
10^{-14}	2.718281828459045	0.0
10^{-15}	2.718281828459045	0.0

Tableau 1.2 – Valeurs de $f'(x)$ calculées par le pas complexe

ont une partie réelle et une partie imaginaire. Les nombres hyperduaux ont une partie réelle, une partie ϵ_1, une partie ϵ_2 et une partie $\epsilon_1\epsilon_2$ définies de sorte que :

$$\epsilon_1^2 = \epsilon_2^2 = (\epsilon_1\epsilon_2)^2 = 0, \tag{1.5}$$

$$\epsilon_1 \neq 0; \epsilon_2 \neq 0; (\epsilon_1\epsilon_2) \neq 0, \tag{1.6}$$

$$\epsilon_1\epsilon_2 = \epsilon_2\epsilon_1. \tag{1.7}$$

Prenons $x \in \mathbb{R}$ (sa représentation hyperduale est donc $x = x + 0 \cdot \epsilon_1 + 0 \cdot \epsilon_2 + 0 \cdot \epsilon_1\epsilon_2$) et h défini par :

$$h = h_1\epsilon_1 + h_2\epsilon_2 + 0\epsilon_1\epsilon_2. \tag{1.8}$$

Le développement de Taylor autour de x d'une fonction $f \in \mathcal{C}^2$ ayant comme pas h est donné par :

$$f(x+h) = f(x) + h_1 f'(x)\epsilon_1 + h_2 f'(x)\epsilon_2 + h_1 h_2 f''(x)\epsilon_1\epsilon_2. \tag{1.9}$$

En effet, les propriétés de ϵ_1, ϵ_2 et $\epsilon_1\epsilon_2$ font en sorte que tous les termes d'ordre trois et plus de la série de Taylor sont nuls. En isolant les parties ϵ_1, ϵ_2 ($\text{part}_{\epsilon_1}, \text{part}_{\epsilon_2}$) et la partie $\epsilon_1\epsilon_2$ ($\text{part}_{\epsilon_1\epsilon_2}$) de $f(x+h)$ on peut extraire $f'(x)$ et $f''(x)$:

$$f'(x) = \frac{\text{part}_{\epsilon_1}(f(x+h))}{h_1} = \frac{\text{part}_{\epsilon_2}(f(x+h))}{h_2},$$
$$f''(x) = \frac{\text{part}_{\epsilon_1\epsilon_2}(f(x+h))}{h_1 h_2}.$$

Donc, de la même manière qu'il est possible d'isoler la partie imaginaire d'un nombre complexe, il est possible d'isoler les parties ϵ_1, ϵ_2 et $\epsilon_1\epsilon_2$ d'un nombre hyperdual. Utilisant (1.9), les dérivées et dérivées secondes de f sont faciles à extraire, en particulier si

$h_1 = h_2 = 1$. Donc, contrairement aux différences finies et à la méthode du pas complexe, il n'est pas nécessaire de prendre h petit.

On peut faire une extension pour les fonctions $f : \mathbb{R}^n \to \mathbb{R}$ et construire le gradient et la hessienne. Soit e_i la i-ème colonne de la matrice identité. On peut construire le gradient et la hessienne en même temps :

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\text{part}_{\epsilon_1\epsilon_2}[f(x + h_1\epsilon_1 e_i + h_2\epsilon_2 e_j + 0\epsilon_1\epsilon_2)]}{h_1 h_2},$$
$$\frac{\partial f(x)}{\partial x_i} = \frac{\text{part}_{\epsilon_1}[f(x + h_1\epsilon_1 e_i + h_2\epsilon_2 e_j + 0\epsilon_1\epsilon_2)]}{h_1}.$$

En effet, il suffit de calculer $f(x + h_1\epsilon_1 e_i + h_2\epsilon_2 e_j + 0\epsilon_1\epsilon_2)$ et d'extraire directement l'information d'ordre un et deux. Nous verrons dans la Section 2.2.2 comment les nombres hyperduaux peuvent être utilisés pour avoir de l'information d'ordre trois.

1.4.3 Différentiation automatique

La différentiation automatique (DA) est une technique pour calculer des dérivées de manière exacte. Une manière de faire de la DA est d'utiliser le graphe d'évaluation de la fonction. Cette technique fonctionne de deux manières : le mode direct et le mode inverse. Une façon de visualiser ces modes est de regarder la manière dont le graphe d'évaluation de la fonction est parcouru. L'idée de base de la DA est de transformer la chaîne de calculs du graphe d'évaluation de la fonction en une autre chaîne de calculs qui nous permet de calculer le gradient de la fonction. La règle d'enchaînement est un outil important de la DA, elle s'énonce ainsi :

$$\Big(f\Big(g(x)\Big)\Big)' = f'\Big(g(x)\Big) \cdot g'(x),$$
$$\nabla\Big(f\Big(g(x)\Big)\Big) = \Big(\nabla f(g(x))\Big) \cdot \nabla g(x).$$

Une présentation de la DA et de ses deux modes dans le cadre de l'optimisation est disponible dans [Dus08]. On peut retrouver une présentation plus exhaustive dans [GW08]. La DA permet de borner le coût du gradient en fonction du coût de la fonction objectif. En effet, calculer la fonction objectif a un certain coût. Ce coût peut être représenté par exemple par le temps de calcul de la fonction. Posons $c = \text{coût}(\nabla f(x))$, dans [BS83], les auteurs stipulent qu'avec des bons outils de DA, $c \leqslant 3 \cdot \text{coût}(f(x))$. Autrement dit, si calculer $f(x)$ prend une seconde, calculer le gradient ne devrait pas prendre plus de trois secondes. Griewank démontre qu'une implémentation du mode inverse assure que dans le pire des cas on a $c \leqslant 5 \cdot \text{coût}(f(x))$ [GW08]. Ces bornes sont valables lorsqu'on utilise le mode inverse de la différentiation automatique et que le graphe d'évaluation de notre fonction est à portée de main. Nous présentons un bref exemple de l'utilisation de la DA et de ses modes inverse et directe.

Prenons la fonction de Rosenbrock :

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2. \tag{1.10}$$

On note que le minimum global de la fonction de Rosenbrock est $x^* = (1, 1)$. On peut obtenir $f(x)$ en construisant des fonctions y_i intermédiaires. Les fonctions y_1, y_2 et y_7 permettent d'extraire les composantes x_1 et x_2 de x. Ces fonctions intermédiaires sont construites telles que leurs dérivées soient faciles à calculer, c'est-à-dire des fonctions élémentaires dont la dérivée possède une formule analytique connue : des polynômes, des additions de polynômes ou des fonctions trigonométriques par exemple (ce n'est pas nécessaire dans cet exemple). En utilisant la règle d'enchaînement et les fonctions y_i il

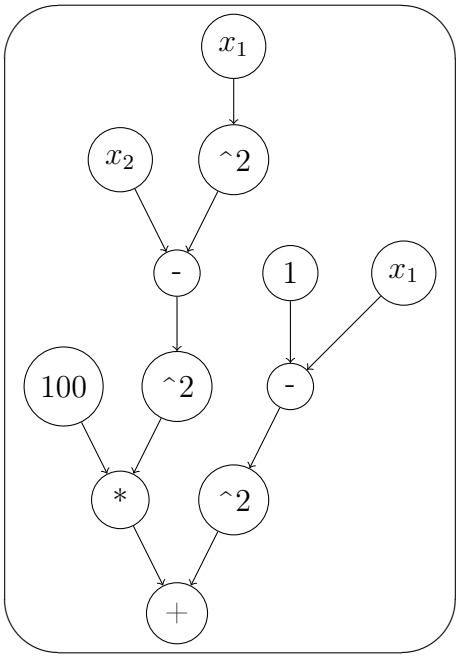

Figure 1.1 – Graphe acyclique orienté de la fonction de Rosenbrock

est possible de construire le gradient de f. Les résultats de la DA utilisant le mode direct sont consignés dans le tableau 1.3.

La figure 1.1 présente le graphe acyclique orienté (GAO) de la fonction de Rosenbrock. On peut y voir les scalaires x_1 et x_2 ainsi que les opérations élémentaires nécessaires pour passer de x_1 et x_2 à $f(x)$. Ce graphe est intimement lié aux fonctions y_i définies ci-haut.

Le mode inverse fonctionne autrement. On utilise les quantités adjointes définies par (1.11) et obéissant à la relation de récurrence (1.12)

$$y_i^* = \frac{\partial f}{\partial y_i}, \tag{1.11}$$

$$y_j^* = \sum \frac{\partial f_i}{\partial y_j} y_i^*. \tag{1.12}$$

On peut expliciter les valeurs adjointes du mode inverse de la DA dans le tableau 1.4.

y_i	y_i	∇y_i	∇y_i
$y_1 = (1\ \ 0)^T x$	x_1	$\nabla y_1 = (1\ \ 0)^T$	$(1\ \ 0)^T$
$y_2 = (0\ \ 1)^T x$	x_2	$\nabla y_2 = (0\ \ 1)^T$	$(0\ \ 1)^T$
$y_3 = y_1^2$	x_1^2	$\nabla y_3 = 2y_1 \nabla y_1$	$(2x_1\ \ 0)^T$
$y_4 = y_2 - y_3$	$x_2 - x_1^2$	$\nabla y_4 = \nabla y_2 - \nabla y_3$	$(-2x_1\ \ 1)^T$
$y_5 = y_4^2$	$(x_2 - x_1^2)^2$	$\nabla y_5 = 2y_4 \nabla y_4$	$(-4x_1(x_2 - x_1^2)\ \ 2(x_2 - x_1^2))^T$
$y_6 = 100y_5$	$100(x_2 - x_1^2)^2$	$\nabla y_6 = 100 \nabla y_5$	$(-400x_1(x_2 - x_1^2)\ \ 200(x_2 - x_1^2))^T$
$y_7 = (1\ \ 0)^T x$	x_1	$\nabla y_7 = (1\ \ 0)^T$	$(1\ \ 0)^T$
$y_8 = 1 - y_7$	$1 - x_1$	$\nabla y_8 = -\nabla y_7$	$(-1\ \ 0)^T$
$y_9 = y_8^2$	$(1 - x_1)^2$	$\nabla y_9 = 2y_8 \nabla y_8$	$(-2(1 - x_1)\ \ 0)^T$
$f = y_6 + y_9$	$f(x)$	$\nabla f = \nabla y_6 + \nabla y_9$	$\begin{pmatrix} -400x_1(x_2 - x_1^2) - 2(1 - x_1)^2 \\ 200(x_2 - x_1^2) \end{pmatrix}$

Tableau 1.3 – Étapes du mode direct de la DA pour la fonction de Rosenbrock

Avec le mode inverse de la DA on peut borner le coût de calcul de $\nabla f(x)$ par 5 fois le coût de calcul de $f(x)$ [GW08]. Ce résultat est remarquable, car il signifie que le coût du gradient n'est pas borné par n, mais par le coût de $f(x)$ (qui lui, peut dépendre de n). Nous verrons comment obtenir de l'information d'ordre 3 à partir de la DA dans la Section 2.2.1.

	Adjointes	Valeurs
0	$x^* = 0$	$(0 \quad 0)$
1	$f^* = 1$	1
2	$y_9^* = f^*$	1
3	$y_8^* = y_9^*(2y_8)$	$2(1 - x_1)$
4	$y_7^* = -y_8$	$-2(1 - x_1)$
5	$x^* += y_7^*(1 \quad 0)^T$	$(-2(1 - x_1) \quad 0)^T$
6	$y_6^* = f^*$	1
7	$y_5^* = 100y_6^*$	100
8	$y_4^* = y_5^*(2y_4)$	$200(x_2 - x_1^2)$
9	$y_3^* = -y_4^*$	$-200(x_2 - x_1^2)$
10	$y_2^* = y_4^*$	$200(x_2 - x_1^2)$
11	$y_1^* = y_3^*(2y_1)$	$-400x_1(x_2 - x_1^2)$
12	$x^* += y_2^*(1 \quad 0)^T$	$\begin{pmatrix} -2(1 - x_1) - 400x_1(x_2 - x_1^2) \\ 0 \end{pmatrix}$
13	$x^* += y_1^*(0 \quad 1)^T$	$\begin{pmatrix} -2(1 - x_1) - 400x_1(x_2 - x_1^2) \\ 200(x_2 - x_1^2) \end{pmatrix}$

Tableau 1.4 – Quantités adjointes de la fonction de Rosenbrock calculées dans le mode inverse de la DA

CHAPITRE 2

Convergence locale

Dans cette section nous voulons résoudre (1) à partir d'un voisinage d'une solution isolée. Le schéma de base pour résoudre un tel problème est le suivant :

Algorithme 2 : Algorithme de descente

Données : x_0 dans un voisinage de la solution, $\epsilon > 0$ et $M < \infty$;

$x \leftarrow x_0$, iter$\leftarrow$0 ;
tant que *(*$\|\nabla f(x)\| \geqslant \epsilon$ *et iter* $< M$*)* **faire**
    $d \leftarrow$ direction de descente ;
    $x \leftarrow x + d$;
    iter $\leftarrow$ iter $+ 1$;
retourner x

Nous remarquons que le critère d'arrêt $\|\nabla f(x)\| < \epsilon$ permet d'approximer un point stationnaire. L'étape la plus difficile dans ce schéma est de calculer une direction de descente d, c'est-à-dire une direction d telle que $\nabla f(x)^T d < 0$. Il existe plusieurs directions de descente, mais certaines permettent de converger plus rapidement vers une solution. Nous allons présenter différentes directions utilisées ainsi que des preuves pour justifier que l'utilisation de ces directions dans l'algorithme 2 assure la convergence vers un point stationnaire. La terminologie « méthode » est utilisée pour désigner les différents choix

de direction, par exemple la méthode de Newton correspond à utiliser la direction de Newton dans l'algorithme 2.

Un aspect important pour comparer les différents choix de direction d est la vitesse de convergence asymptotique. Le choix de d dans l'algorithme 2 influence la vitesse à laquelle une suite d'itérés $\{x_k\}$ converge vers la solution. Dans le contexte de la convergence locale, on suppose qu'un point d'accumulation x^* satisfait aux conditions suffisantes. Dans ce cas, c'est un minimum local qui attire les itérés et toute la suite $\{x_k\}$ converge vers x^*.

Nous utilisons la définition de convergence asymptotique définie dans [Dus15]. Nous notons $\epsilon_k = x_k - x^*$ l'erreur absolue entre un itéré x_k et la solution x^*.

Définition 2.1 *La vitesse de convergence de la suite $\{x_k\}$ vers x^*, telle que pour tout k $x_k \neq x^*$, s'exprime à l'aide des scalaires p et γ dans l'expression suivante*

$$\limsup_{k\to\infty} \frac{|\epsilon_{k+1}|}{|\epsilon_k|^p} = \gamma < \infty. \tag{2.1}$$

L'ordre de convergence de la suite $\{x_k\}$ est donné par la plus grande valeur que p puisse prendre pour que la limite ci-haut demeure finie. Lorsque $p = 1$, γ est nommé taux de convergence et on doit avoir $\gamma < 1$.

Autrement dit, plus p est grand, plus $\{x_k\}$ converge rapidement vers x^*, du moment que x_0 est dans un certain voisinage de x^*. Lorsque $p = 1$, on parle de convergence linéaire, si $p = 2$, on parle de convergence quadratique (i.e. $|\epsilon_{k+1}| \sim \mathcal{O}\left(|\epsilon_k|^2\right)$) et si $p = 3$, on parle de convergence cubique (i.e. $|\epsilon_{k+1}| \sim \mathcal{O}\left(|\epsilon_k|^3\right)$). Bien souvent, il est impossible de calculer ϵ_k, car la valeur de x^* n'est pas disponible. Pour observer les différents ordres de convergence des différentes méthodes que nous présentons dans ce chapitre nous regardons la suite $\{\|\nabla f(x_k)\|\}$. Un algorithme a un ordre de convergence p si $\|\nabla f(x_{k+1})\| \sim \mathcal{O}(\|\nabla f(x_k)\|^p)$. De plus, si x^* satisfait aux conditions suffisantes, alors $\|\nabla f(x_k)\| \sim \|x_k - x^*\|$. Autrement dit, $\nabla f(x_k)$ converge à la même vitesse que ϵ.

Finalement, un autre aspect important pour comparer différentes méthodes est la quantité d'opérations élémentaires (+,-,*, /) nécessaires pour calculer d. Si une méthode a une convergence asymptotique cubique, mais qu'elle nécessite beaucoup d'opérations élémentaires par itération (donc beaucoup de temps de calcul) cette méthode peut être moins intéressante qu'une méthode qui a une convergence asymptotique plus faible mais qui atteint une solution plus rapidement en termes de temps de calcul.

2.1 Direction de Newton

La direction de Newton d_N est directement issue de la fonction $q_x(d)$ définie en 1.1, il s'agit de la direction qui minimise q_x si celle-ci est convexe. Si $q_x(d)$ n'est pas convexe, elle est non bornée inférieurement.

Le minimum de q_x a été calculé dans la Section 1.2, d'où la Définition 1.3 de la direction de Newton.

En pratique, $\nabla^2 f(x)$ n'est jamais inversée, car il s'agit d'une procédure très coûteuse et il n'est pas garanti que l'inverse existe. Dans nos travaux, nous utilisons principalement les décompositions LDL^T ou BBK (*bounded Bunch-Kaufman*) pour résoudre 1.3. Il est intéressant d'utiliser une décomposition, car elle peut être réutilisée pour calculer les directions d'ordre supérieur.

La direction de Newton assure la convergence quadratique si on l'utilise dans l'Algorithme 2. Il existe plusieurs façons de démontrer la convergence quadratique de la méthode de Newton. Nous présentons le théorème 2.2 dont la preuve est disponible dans [Fle87].

Théorème 2.2 *Supposons que $f \in C^2$ et que la hessienne de f est Lipschitz, i.e. il existe $\lambda > 0$ tel que $\|\nabla^2 f(x) - \nabla^2 f(y)\| \leqslant \lambda \|x - y\|$ pour tout x, y dans un voisinage*

d'un minimum local x^*. Soit $\{x_k\}$ la suite d'itérés générés par l'algorithme 2. Si x_0 est suffisamment proche de x^* et si $\nabla^2 f(x^*)$ est définie positive, alors la méthode de Newton est bien définie pour tout k et converge de manière quadratique vers la solution.

Le dernier aspect important à considérer est le nombre d'opérations élémentaires nécessaires pour calculer la direction de Newton. Les opérations élémentaires sont les additions, soustractions, multiplications et divisions. Le nombre d'opérations élémentaires pour chaque étape du calcul de la direction de Newton est consigné dans le tableau 2.1.

Opérations	Coût (op. élémentaires)
Décomposition LDL^T	$\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{5}{2}$
Résoudre le système linéaire $\nabla^2 f(x)d = -\nabla f(x)$	$2n^2 - n$
Obtenir $\nabla^2 f$	nc
Coût de ∇f	c
Soustraction de vecteurs	n
Total	$\frac{1}{3}n^3 + \frac{5}{2}n^2 - \frac{5}{6}n + (n+1)c$

Tableau 2.1 – Nombre d'opérations élémentaires de la méthode de Newton

On remarque que le coût pour calculer un gradient n'est qu'une simple constante c bornée par le coût de calcul de $f(x)$ tel que discuté à la Section 1.4.3. On note que $f(x)$ n'est jamais calculé dans l'algorithme 2, mais son gradient l'est.

2.2 Méthodes de Halley, SuperHalley et Chebyshev

Dans ce document, nous définissons les méthodes d'ordre supérieur comme les méthodes qui offrent une convergence asymptotique strictement supérieure à deux. Autrement dit, les méthodes dont la convergence asymptotique est supérieure à celle de Newton sont considérées comme des méthodes d'ordre supérieur. Pour atteindre un tel ordre de conver-

gence asymptotique, certaines méthodes utilisent les dérivées d'ordre trois. L'information des dérivées d'ordre un et deux est encodée dans un vecteur et une matrice respectivement. On note $\nabla^3 f(x)$ le tenseur d'ordre 3 des dérivées troisièmes de f. Une matrice (ou tenseur de dimension 2) peut être représentée par un carré, tandis qu'un tenseur d'ordre 3 peut être vu comme un cube. Une représentation possible de $\nabla^3 f$ serait donc un « cube » $n \times n \times n$ où $\nabla^3 f_{(i,j,k)} = \frac{\partial^3 f}{\partial x_i \partial x_j \partial x_k}$. Au même titre que le résultat d'une multiplication matrice-vecteur est un vecteur, le résultat d'une multiplication tenseur-vecteur est une matrice. Le résultat d'un produit tenseur-vecteur-vecteur est donc un vecteur.

Pour $\alpha \in \mathbb{R}$, le schéma itératif (2.2) résume différentes méthodes d'ordre supérieur qui utilisent des dérivées d'ordre trois pour obtenir une convergence locale cubique (où d_N est définie par (1.3)).

$$\begin{cases} x_{k+1} = x_k - \left[I + \frac{1}{2}L(x_k)(I - \alpha L(x_k))^{-1}\right] \nabla^2 f(x_k)^{-1} \nabla f(x_k), \\ \\ L(x) = \nabla^2 f(x)^{-1} \nabla^3 f(x) \nabla^2 f(x)^{-1} \nabla f(x) = -\nabla^2 f(x)^{-1} \nabla^3 f(x) d_N. \end{cases} \tag{2.2}$$

On note que $L(x)$ est une matrice. On se permet de supposer que $\nabla^2 f(x)^{-1}$ existe.

Dans (2.2), le choix de α distingue les différentes méthodes :

- $\alpha = 0$: méthode de Chebyshev ;
- $\alpha = \frac{1}{2}$: méthode de Halley ;
- $\alpha = 1$: méthode de SuperHalley.

On peut écrire les directions de Chebyshev (d_C) et Halley (d_H) explicitement ainsi :

$$\begin{aligned} &\nabla f(x_k) + \nabla^2 f(x_k) d_N = 0 && \iff \nabla^2 f(x_k) d_N = -\nabla f(x_k), \\ &\nabla f(x_k) + \nabla^2 f(x_k) d_C + \tfrac{1}{2}\nabla^3 f(x_k) d_N d_N = 0 && \iff \nabla^2 f(x_k) d_C = -\nabla f(x_k) - \tfrac{1}{2}\nabla^3 f(x_k) d_N d_N, \\ &\nabla f(x_k) + \nabla^2 f(x_k) d_H + \tfrac{1}{2}\nabla^3 f(x_k) d_N d_H = 0 && \iff (\nabla^2 f(x_k) + \tfrac{1}{2}\nabla^3 f(x_k) d_N) d_H = -\nabla f(x_k). \end{aligned}$$

L'expression explicite de la direction de SuperHalley (d_{SH}) est la suivante (la direction

d_V est un calcul intermédiaire pour simplifier le calcul de d_{SH}) :

$$\big(\nabla^2 f(x_k) + \nabla^3 f(x_k) d_N\big) d_V = -\nabla f(x_k),$$
$$\nabla^2 f(x_k) d_{SH} = -\big(\nabla f(x_k) + \tfrac{1}{2}\nabla^3 f(x_k) d_N d_V\big).$$

Pour trouver la direction de Chebyshev, le système linéaire à résoudre utilise la matrice hessienne. L'information des dérivées d'ordre supérieur est utilisée dans le terme de droite du système linéaire. Donc, si la direction de Newton a été obtenue avec une factorisation de la hessienne, cette factorisation peut être réutilisée pour trouver la direction de Chebyshev. Donc le coût supplémentaire de la direction de Chebyshev est uniquement dans le calcul des dérivées d'ordre supérieur.

Pour calculer les directions de Halley et d_V (qui est utilisée dans la direction de Super-Halley), la décomposition de la hessienne ne peut pas être réutilisée pour résoudre les nouveaux systèmes linéaires. En effet, l'information des dérivées d'ordre trois est utilisée dans le terme de gauche du système linéaire. Pour ces directions il faut donc factoriser une nouvelle matrice en plus de calculer des dérivées d'ordre supérieur.

Obtenir de l'information d'ordre trois peut être coûteux, car on doit former un tenseur (n^3 éléments à garder en mémoire). Le tenseur d'une fonction $f : \mathbb{R}^{1000} \to \mathbb{R}$ a un milliard d'éléments à garder en mémoire, d'autant plus que les opérations tenseur-matrice et tenseur-vecteur ne sont pas évidentes à définir numériquement. Nous discuterons de ces considérations aux sections 2.2.1 et 2.2.2, mais avant, abordons la convergence de ces méthodes.

Théorème 2.3 *Soit $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ est $\mathcal{C}^3$ dans une boule centré en x^* de rayon ϵ, où x^* est tel que $\nabla f(x^*) = 0$ et $\nabla^2 f(x^*)$ est non singulière.*

Supposons que $\nabla^3 f$ est Lipschitz continue dans une boule ouverte centré en x^* de rayon r. Pour $\alpha \in \{0, \frac{1}{2}, 1\}$ il existe $\epsilon > 0$ tel que pour tout x_0 satisfaisant $\|x_0 - x^*\| < \epsilon$ les

points $\{x_k\}$ générés par le schéma (2.2) sont bien définis. De plus, ils convergent vers x^* avec un ordre de convergence au moins cubique.

Il existe différentes façons de démontrer le théorème 2.3. On peut montrer que ces méthodes sont des cas particuliers de familles de méthodes ayant des propriétés de convergence cubique, ce qui est fait dans [KD14]. Pour démontrer le théorème 2.3 on réfère à [CM90a, CM90b].

En termes d'opérations élémentaires, le coût des directions d'ordre supérieur est plus élevé que celui de la direction de Newton. En effet, les directions d'ordre supérieur d_{HO} (2.2) requièrent la direction de Newton ainsi qu'une correction qui est représentée par

$$d_{\text{HO}} = d_N - \tfrac{1}{2}L(x_k)(I - \alpha L(x_k))^{-1}d_N. \tag{2.3}$$

Pour la direction de Chebyshev $d_C = d_N - \frac{1}{2}\nabla^2 f(x)^{-1}\nabla^3 f(x)d_N d_N$ les coûts supplémentaires par rapport à la méthode de Newton sont consignés dans le tableau 2.2. La raison pour laquelle on utilise une décomposition LDL^T pour résoudre $\nabla^2 f(x)d_N = -\nabla f(x)$ devient donc évidente lorsqu'on regarde le calcul des directions d'ordre supérieur. En effet, si on a une décomposition de la hessienne pour calculer la direction de Newton, on peut réutiliser cette décomposition dans le calcul des directions d'ordre supérieur. Cela nous épargne des opérations élémentaires.

Opérations supplémentaires	Coût (op. élémentaires)
$\nabla^3 f(x)d_N d_N = \nabla\left(\nabla^2 f(x)d_N\right)d_N$	$25c + 30n$
Résoudre un système linéaire	$2n^2 - n$
Addition de vecteurs et produit scalaire	$2n$

Tableau 2.2 – Opérations élémentaires de la méthode de Chebyshev

On peut faire le même travail pour les directions de Halley et SuperHalley. On consigne les coûts de calculs de chaque direction dans le tableau 2.3.

Algo.	Coût (op. élémentaires)	Ordre de convergence asymptotique
Newton	$\frac{1}{3}n^3 + \frac{5}{2}n^2 - \frac{5}{6}n + (n+1)c$	2
Chebyshev	$\frac{1}{3}n^3 + \frac{9}{2}n^2 + \frac{181}{6}n + (n+26)c$	3
Halley	$\frac{2}{3}n^3 + \frac{9}{2}n^2 - \frac{5}{6}n + (n+2)c$	3
SuperHalley	$\frac{2}{3}n^3 + \frac{9}{2}n^2 - \frac{1123}{6}n + (n+151)c$	3

Tableau 2.3 – Coût algébrique des différentes méthodes d'ordre supérieur

2.2.1 Dérivées d'ordre trois par différentiation automatique

La plus grande difficulté du calcul des directions de Chebyshev et (Super)Halley est de calculer efficacement l'information d'ordre trois. En effet, former le tenseur $\nabla^3 f(x)$ est une opération coûteuse (n^3 allocations de mémoire), peu d'outils offrent un calcul efficace des opérations tensorielles et aucun outil ne tire profit de la structure creuse des tenseurs. Nous avons exploré deux méthodes pour calculer les dérivées d'ordre supérieur, et plus exactement, les produits *tenseur-vecteur-vecteur* sans former explicitement le tenseur d'ordre trois. La première méthode repose sur des manipulations d'analyse numérique avec la DA, la seconde utilise les nombres hyperduaux.

On peut utiliser les manipulations d'analyse numérique suivantes pour obtenir l'information des dérivées troisièmes :

$$
\begin{aligned}
&f : \mathbb{R}^n \to \mathbb{R} \\
&\nabla f : \mathbb{R}^n \to \mathbb{R}^n \\
&\phi_u : \mathbb{R}^n \to \mathbb{R} && \text{telle que } \phi_u(x) = \nabla f(x)^T u \\
&\nabla \phi_u : \mathbb{R}^n \to \mathbb{R}^n && \nabla \phi_u(x) = \nabla^2 f(x) u \\
&\psi_v : \mathbb{R}^n \to \mathbb{R} && \text{telle que } \psi_v(x) = \nabla \phi_u(x) v \\
&\nabla \psi_v : \mathbb{R}^n \to \mathbb{R}^n && \nabla \psi_v(x) = \nabla^3 f(x) uv
\end{aligned}
$$

Donc, il nous suffit de poser $u = d_N$ et $v = d_N$ pour calculer la correction $\nabla^3 f(x_k) d_N d_N$

utilisée dans la direction de Chebyshev. Pour calculer le produit $\nabla^3 f(x_k)d_N$ on calcule $\nabla^3 f(x_k)d_N e_i$ (où e_i est la i-ème colonne de la matrice identité) pour $i = 1, ..., n$ ce qui nous donne la i-ème colonne de la matrice $\nabla^3 f(x_k)d_N$.

Pour calculer les gradients des fonctions f, ϕ et ψ de manière exacte on utilise la différentiation automatique (DA).

En théorie, avec des outils de DA idéaux, cette méthode de calcul de $\nabla^3 f(x)uv$ devrait être relativement peu coûteuse. En effet, le coût de calcul de $\nabla^3 f(x)uv$ ne devrait pas excéder 125 (ou 5^3) fois le coût de la fonction objectif $f(x)$ en pire cas. Avec des outils de DA idéaux cette constante est une surestimation. Ce résultat est remarquable, car il implique que le coût de $\nabla^3 f(x)uv$ est indépendant de n, mais seulement dépendant du coût de f.

Cependant, il y a de nombreuses considérations relatives au code qui rendent cette borne difficile à atteindre et rendent le calcul de $\nabla\psi_v(x) = \nabla^3 f(x)uv$ coûteux. Tout d'abord, cette borne suppose que l'on définit le graphe d'évaluation de f une seule fois et qu'on le réutilise. Il est possible de faire ainsi, mais les outils de DA à notre disposition ne nous permettent pas de calculer ϕ ou ψ si on utilise le même graphe d'évaluation de f d'une itération à l'autre. Donc, on doit calculer le graphe d'évaluation de f à chaque itération. On doit également le faire pour les fonctions ϕ et ψ. Recalculer le graphe d'évaluation de trois fonctions est extrêmement coûteux.

Un autre problème avec cette méthode est qu'il est impossible avec les outils à notre disposition de travailler avec des matrices/tenseurs creux. De nombreuses fonctions sont telles que leur hessienne $\nabla^2 f(x)$ a une grande proportion de zéros. On dit que ce sont des matrices creuses. Il est avantageux, lorsque n est grand, d'exploiter ce genre de structure dans nos matrices. Le calcul de nombreux zéros récurrents est lourd et inutile.

L'accumulation de ces problèmes fait en sorte que le coût de $\nabla^3 f(x)uv$ dépend de la

valeur de n et devient rapidement très coûteux en temps de calcul ; il est donc très difficile de tirer profit de la convergence asymptotique cubique. Un outil idéal de DA nous permettrait de définir les graphes d'évaluation de f, ϕ et ψ et de les mettre à jour au fur et à mesure des itérations en plus de tirer profit de la structure creuse de nombreux problèmes.

2.2.2 Dérivées d'ordre trois par nombres hyperduaux

Les nombres hyperduaux sont une autre option pour obtenir de l'information des dérivées troisièmes sans former explicitement $\nabla^3 f(x)$. On peut utiliser la série de Taylor du gradient pour obtenir le produit $\nabla^3 f(x)uv$. On prend x, u et v des vecteurs, de tel sorte que $x_{hd} = x + \epsilon_1 u + \epsilon_2 v + 0\epsilon_1\epsilon_2$ est un vecteur où chaque élément est un nombre hyperdual (Section 1.4.2) :

$$\nabla f(x + \epsilon_1 u + \epsilon_2 v + 0\epsilon_1\epsilon_2) = \nabla f(x) + \epsilon_1 \nabla^2 f(x)u + \epsilon_2 \nabla^2 f(x)v + \epsilon_1\epsilon_2 \nabla^3 f(x)uv.$$

Cette nouvelle idée d'utiliser les nombres hyperduaux pour calculer le produit tenseur-vecteur-vecteur est très intéressante : il n'y a qu'un seul gradient à calculer. Il n'y a pas de gradient imbriqué dans d'autres gradients. Cependant, calculer le gradient d'un nombre hyperdual n'est actuellement pas possible à moins de calculer « manuellement » le gradient de la fonction objectif. On ne peut pas utiliser les outils de DA actuels pour calculer le gradient d'une fonction avec un nombre hyperdual. Il faudrait faire des modifications de ces outils ou en créer de nouveau. Il s'agit d'un défi intéressant, mais qui déborde du cadre de ce projet.

Étant donné que les outils de DA disponibles n'offrent pas encore les fonctionnalités dont nous avons besoin et que le calcul de dérivées troisièmes est très coûteux, nous limiterons nos tests numériques des méthodes de Chebyshev et SuperHalley à des fonctions ayant

moins de 100 variables. Les méthodes d'ordre supérieur seront pénalisées en termes de temps de calcul, mais on pourra illustrer leur rapidité en termes de nombre d'itérations.

2.3 Méthode de Shamanskii-Traub

La méthode de Shamanskii offre également la convergence cubique asymptotique, mais elle n'utilise que les dérivées d'ordre un et deux. L'idée de la méthode de Shamanskii est la suivante : proche d'une solution x^* la hessienne ne devrait pas être très différente d'une itération à l'autre ($\nabla^2 f(x_k) \approx \nabla^2 f(x_{k+1})$). Donc, si une factorisation de la hessienne est disponible, elle peut être utilisée pour résoudre un deuxième système linéaire. La méthode de Shamanskii peut être représentée par le schéma suivant :

$$\begin{aligned} \nabla^2 f(x_k) d_N &= -\nabla f(x_k) \\ x_{\tilde{S}} &= x_k + d_N \\ \nabla^2 f(x_k) d_{\tilde{S}} &= -\nabla f(x_{\tilde{S}}) \\ x_{k+1} = x_{\tilde{S}} + d_{\tilde{S}} &= x_k + d_N + d_{\tilde{S}}. \end{aligned}$$

Nous notons la direction de Shamanskii $d_S = d_N + d_{\tilde{S}}$. On remarque que, tout comme pour les directions de Chebyshev, Halley et SuperHalley, la direction de Shamanskii consiste à ajouter une correction à la direction de Newton. Il est préférable de résoudre le premier système linéaire à l'aide d'une décomposition, afin de pouvoir l'utiliser de nouveau pour résoudre le deuxième système linéaire. Pour prouver la convergence cubique de la méthode de Shamanskii, on résume l'itération de Shamanskii en une seule ligne (en supposant que l'inverse de la hessienne existe) :

$$x^{k+1} = x^k - \nabla F(x^k)^{-1}[F(x^k) + F(x^k - \nabla F(x^k)^{-1} F(x^k))] \tag{2.4}$$

où $F = \nabla f$ et $\nabla F = \nabla^2 f$.

Théorème 2.4 *Soit $F : \mathbb{R}^n \to \mathbb{R}^n$ différentiable dans une boule ouverte S centrée en x^* (solution de (1)) de rayon δ telle que :*

$$\|\nabla F(x) - \nabla F(x^*)\| \leqslant \gamma \|x - x^*\|, \forall x \in S.$$

De plus, supposons que $F(x^*) = 0$ et que $\nabla F(x^*)$ est non singulière. Alors x^* est un point d'accumulation de l'itération de Shamanskii défini par (2.4) et son taux de convergence est cubique.

Démonstration. On suppose que $\nabla F(x^*)$ est non singulière, donc inversible. On sait que l'itération de Newton converge de façon quadratique. Donc si on pose $x_N = x - \nabla F(x)^{-1} F(x)$ le point donné par l'itération de Newton on peut écrire :

$$\|x_N - x^*\| \leqslant \eta \|x - x^*\|^2$$

où x est dans une boule S_1 centrée en x^* et de rayon δ_1 telle que $S_1 \subset S$. Donc, l'application suivante est bien définie sur $S_2 = \bar{S}(x^*, \delta_2) \subset S_1$ où $\delta_2^2 \leqslant \frac{\delta_1}{\eta}$ pour un certain $\eta > 0$:

$$G(x) = x_N - \nabla F(x)^{-1} F(x_N). \tag{2.5}$$

Donc, si $\|\nabla F(x)^{-1}\| \leqslant \beta$ pour tout $x \in S_2$ alors :

$$\begin{aligned}
\|G(x) - x^*\| &\leqslant \|\nabla F(x)^{-1}\| \|\nabla F(x) x_N - F(x_N) - \nabla F(x) x^*\| \\
&\leqslant \beta \|F(x_N) - F(x^*) - \nabla F(x^*)(x_N - x^*)\| + \beta \|\nabla F(x^*) - \nabla F(x)\| \|x_N - x^*\| \\
&\leqslant \tfrac{1}{2} \beta \gamma \|x_N - x^*\|^2 + \beta \gamma \|x - x^*\| \|x_N - x^*\| \\
&\leqslant [\beta \gamma \eta (\tfrac{1}{2} \eta \delta_2 + 1)] \|x - x^*\|^3
\end{aligned}$$

pour tout $x \in S_2$. D'où la convergence cubique. ■

Finalement, notons qu'il est possible d'atteindre une convergence asymptotique d'ordre p avec la méthode de Shamanskii si on répète p fois le processus avec la même hessienne. Le

problème est que le voisinage autour de la solution pour observer un tel phénomène décroît lorsque p augmente, ce qui rend cette idée théoriquement intéressant, mais d'intérêt limité en pratique.

La méthode de Shamanskii pourrait être attribuée à Traub [Tra64], une nomenclature plus appropriée serait la méthode de « Shamanskii-Traub ». Par exemple, dans [OR00] les notations de Traub et Shamanskii sont utilisées. D'autres auteurs utilisent *composite Newton methods* ou *Newton with periodic hessian evaluations*. Nous nous contentons d'utiliser l'appellation « méthode de Shamanskii ».

2.4 Méthodes quasi-Newton

Une présentation exhaustive des méthodes de quasi-Newton peut être trouvés dans [DJM74]. Nous faisons ici un rappel des méthodes qui seront utilisées.

Un des principaux désavantages de la méthode de Newton est le calcul de $\nabla^2 f(x_k)$ qui correspond à $\mathcal{O}(n^2)$ évaluations de fonctions scalaires à chaque itération. Le but des méthodes quasi-Newton est d'approximer $\nabla^2 f(x_k)$ ou $\nabla^2 f(x_k)^{-1}$. On note B_k une matrice qui est une approximation de $\nabla^2 f(x_k)$ et H_k une approximation de $\nabla^2 f(x_k)^{-1}$. Ces matrices sont utilisées dans le schéma itératif (2.6), similaire à celui de Newton.

$$x_{k+1} = x_k - H_k \nabla f(x_k). \tag{2.6}$$

Les matrices B_k et H_k doivent respecter certaines propriétés :

- elles doivent être symétriques ; la hessienne est symétrique et il s'agit d'une propriété qui doit être partagée par ses approximations ;
- $B_{k+1}s = y$ où $y = \nabla f(x_{k+1}) - \nabla f(x)$ et $s = x_{k+1} - x_k$; ;
- $-B_k^{-1}\nabla f(x_k)$ (ou $-H_k \nabla f(x_k)$) doit être une direction de descente qui ressemble

à la direction de Newton pour garder les propriétés de convergence superlinéaire de la méthode de Newton ;

— si B_k est définie positive, alors B_{k+1} doit l'être également.

Nous présentons les formules de Davidon-Fletcher-Powell (DFP) et de Broyden-Fletcher-Goldfarb-Shanno (BFGS) qui approximent la hessienne, ou une matrice H telle que $H^{-1} = \nabla^2 f(x)$, et qui respectent les propriétés énumérées plus haut.

2.4.1 Itération DFP

La formule de la mise à jour DFP est la suivante :

$$B_{k+1} = B_k + \frac{(y - B_k s)^T y + y(y - B_k s)^T}{y^T s} \tag{2.7}$$

$$= (I - \frac{ys^T}{y^T s}) B_k (I - \frac{sy^T}{y^T s}) + \frac{yy^T}{y^T s}. \tag{2.8}$$

On résume les propriétés de (2.7) dans le théorème 2.5.

Théorème 2.5 *Soit B_k une matrice $n \times n$ non-singulière et symétrique. Soit B_{k+1} définie par (2.7) ou (2.8) et s et y dans $\mathbb{R}^n$ tels que $y^T s \neq 0$. Alors B_{k+1} est non singulière si, et seulement si, $y^T B^{-1} y \neq 0$. Si B_{k+1} est non singulière alors $H_{k+1} = B_{k+1}^{-1}$ est définie par*

$$H_{k+1} = H_k + \frac{ss^T}{s^T y} - \frac{Hyy^T H}{y^T H y}. \tag{2.9}$$

De plus, si B_k est définie positive, alors B_{k+1} est définie positive si, et seulement si, $y^T s > 0$.

La preuve de ce théorème est dans [DJM74]. Il est également possible de prouver que la formule de DFP respecte les propriétés énumérées plus haut, et donc, que la formule DFP est une bonne approximation de la hessienne.

2.4.2 Itération BFGS

Une autre formule de mise à jour de la hessienne est la formule de BFGS. Cette formule est intimement liée à la formule de DFP et est parfois appelée la formule complémentaire de DFP. Les transformations suivantes permettent de faire le lien entre les formules de DFP et BFGS :

$$s \leftrightarrow y; B_k \leftrightarrow H_k; B_{k+1} \leftrightarrow H_{k+1}.$$

La mise à jour de BFGS est définie par

$$H_{k+1} = (I - \frac{sy^T}{y^T s})H_k(I - \frac{ys^T}{y^T s}) + \frac{ss^T}{y^T s}. \tag{2.10}$$

On résume ses propriétés dans le théorème suivant :

Théorème 2.6 *Soit H_k une matrice $n \times n$ et non-singulière. Soit H_{k+1} définie par (2.10) et y et s tels que $y^T s \neq 0$. Alors H_{k+1} est non-singulière si, et seulement si, $s^T H_k^{-1} s \neq 0$. De plus, si $B_k = H_k^{-1}$ et H_k est non-singulière alors on a que*

$$B_{k+1} = B_k + \frac{yy^T}{y^T s} + \frac{B_k s s^T B}{s^T B_k s}. \tag{2.11}$$

Finalement, si H_k est définie positive, alors H_{k+1} est définie positive si, et seulement si, $y^T s > 0$.

Tout comme pour le théorème 2.5, on peut trouver la preuve du théorème 2.6 dans [DJM74].

2.5 Shamanskii-BFGS

Une idée proposée dans [DJM82] est d'utiliser des mises à jour de la hessienne dans la méthode de Newton pour éviter de calculer la hessienne à chaque itération. Dans

ce document, nous proposons une idée similaire c'est-à-dire combiner la méthode de Shamanskii et la mise à jour de BFGS. Étant donné que l'on réutilise la hessienne, l'idée est de la bonifier d'une mise à jour de BFGS. On présente l'idée dans le schéma suivant où $B_k = \nabla^2 f(x_k)$:

1. Calcul d_N : $B_k d_N = -\nabla f(x_k)$.
2. On pose $s_k = d_N$ et $x_{k_N} = x_k + s_k$.
3. $y_k = \nabla f(x_{k_N}) - \nabla f(x_k)$.
4. $B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{\nabla^2 f(x_k) s_k s_k^T \nabla^2 f(x_k)^T}{s_k^T \nabla^2 f(x_k) s_k}$.
5. Calcul de $\tilde{d}_S$: $B_{k+1}\tilde{d}_S = -\nabla f(x_{k_N})$.
6. $x_{k+1} = x_{k_N} + \tilde{d}_S = x_k + d_N + \tilde{d}_S$.

Donc, on ne construit pas la matrice $\nabla^2 f(x_{k_N})$, on espère que la matrice B_{k+1} l'approxime mieux que $\nabla^2 f(x_k)$. L'idée de construire B_{k+1} est de gagner de la précision, donc possiblement des itérations.

2.6 Exemples numériques

Dans cette section, nous regardons différents résultats numériques qui illustrent les résultats de convergence des différentes méthodes. Tous nos algorithmes ont été codés en Julia [BEKS17]. Par défaut, la majorité des calculs sont toujours fait en `Float64` (double précision). Cela signifie que chaque nombre est encodé dans 64 bits et correspond à une précision numérique maximale de $2.220446049250313 \times 10^{-16}$. Julia offre une représentation des nombres en précision arbitraire, les `BigFloat`. L'avantage de cette représentation est évidemment l'accès à une grande précision. Cependant, étant donné que ce niveau de précision est atteint au niveau logiciel et non au niveau machine, le temps de calcul devient rapidement restrictif en `BigFloat`.

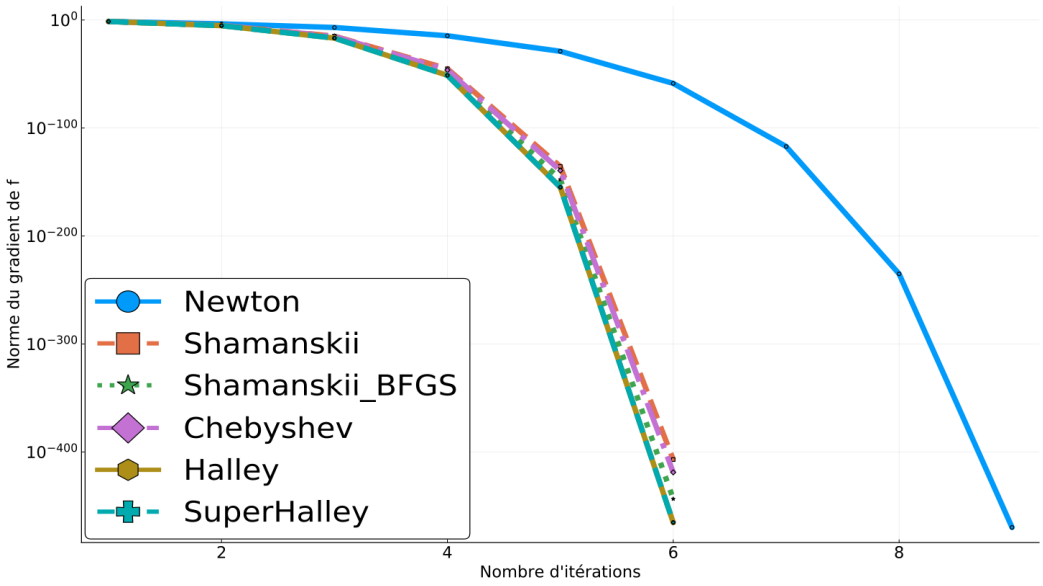

Figure 2.1 – Convergence locale de différentes méthodes appliquées à la fonction de Rosenbrock

Commençons par regarder le comportement des différentes méthodes présentées précédemment sur la fonction de Rosenbrock (1.10). La convergence des différentes méthodes dépend beaucoup du point de départ x_0. Pour nous assurer de la convergence, nous prenons $x_0 = (3.00947, 3.00947)^T$ (pour cet exemple, tous les nombres sont codés en `BigFloat`) afin de démarrer l'algorithme 2 dans un voisinage de la solution. Le seul minimum de la fonction de Rosenbrock est $x^* = (1, 1)$. Dans la Figure (2.1) nous pouvons voir la valeur de $\|\nabla f(x)\|$ au fil des itérations pour les méthodes de Newton, Shamanskii (et sa variante BFGS), Chebyshev et (Super)Halley.

On peut donc apprécier la convergence cubique des méthodes d'ordre supérieur qui nécessitent moins d'itérations pour atteindre une précision de 10^{-400}.

Nous pouvons faire des observations similaires avec la fonction de Wood (2.12) où $x \in \mathbb{R}^4$

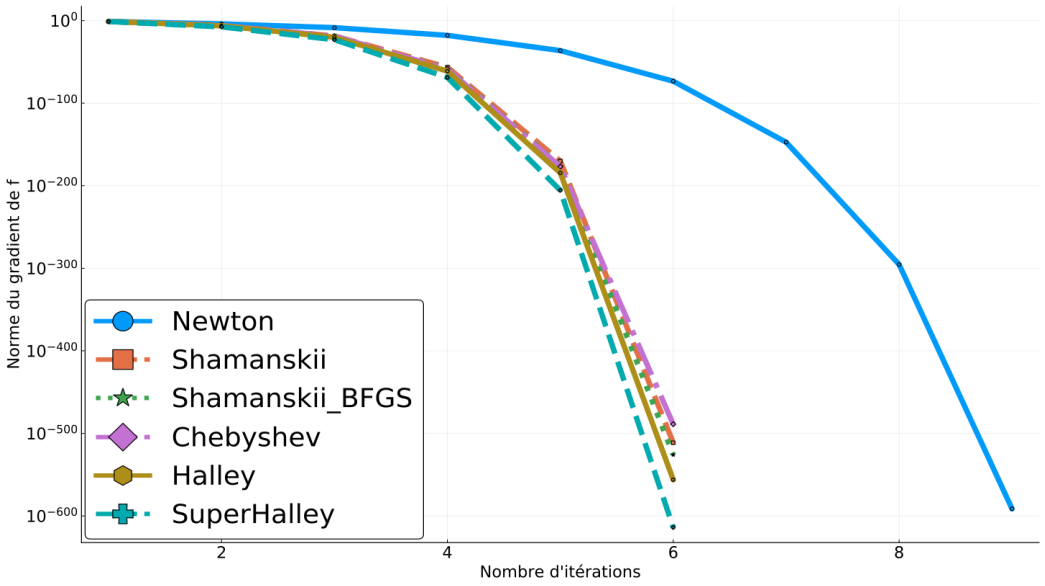

Figure 2.2 – Norme du gradient de la fonction de Wood en fonction du nombre d'itérations

avec $x_0 = (1.00042, 1.00065, 1.00032, 1.00051)^T$.

$$f(x) = 100(x_2 - x_1)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2) + (1 - x_3)^3 + 10(x_2 + x_4 - 2)^2 + 10(x_2 - x_4)^2. \tag{2.12}$$

La Figure (2.2) présente la norme du gradient pour cette fonction et des résultats similaires à ceux obtenus pour la fonction de Rosenbrock sont présentés. La convergence cubique des méthodes d'ordre supérieur est observable.

Cependant, les analyses précédentes des différentes méthodes supposent que l'on a un point x_0 dans un voisinage de la solution. Dans le prochain chapitre, nous nous attardons à globaliser les méthodes, c'est-à-dire trouver un minimum local de f avec un x_0 quelconque.

CHAPITRE 3

Convergence globale

Dans la section précédente, nous avons analysé les propriétés de convergence de différentes méthodes lorsque le point initial x_0 est dans un voisinage de la solution x^*. Nous étudions maintenant des stratégies qui assurent la convergence globale, c'est-à-dire qui assurent que, pour tout point de départ x_0, tout point d'accumulation de la suite engendré par l'algorithme satisfait aux conditions nécessaires d'optimalité si une telle solution existe. Nous notons que nous ne faisons pas la recherche de minima globaux, mais bien la recherche de points satisfaisant aux conditions nécessaires pour un minimum local. Dans notre cas, il s'agit des conditions d'ordre deux.

Il existe de nombreuses stratégies qui garantissent la convergence globale. En premier lieu, nous présentons des stratégies qui, asymptotiquement, se réduisent à la méthode de Newton : la recherche linéaire et les région de confiance. En effet, la méthode de Newton possède des propriétés désirables de convergence quadratique proche d'une solution. Une particularité des stratégies de convergence globale est donc d'utiliser la direction de Newton le plus rapidement possible, c'est-à-dire dès qu'un itéré x_k est dans un voisinage de la solution où la méthode de Newton converge. Par la suite, nous présenterons les

algorithmes ARC qui convergent globalement sans toutefois se réduire à la direction de Newton asymptotiquement.

3.1 Recherche linéaire

Rappelons que l'algorithme 2 consiste à calculer une direction de descente puis se déplacer dans cette direction. Si x_0 est proche d'une solution x^* qui satisfait aux conditions nécessaire de second ordre, ce schéma est suffisant. Cependant, si x_0 est quelconque ce schéma peut produire une suite d'itérés x_k qui :

— ne convergent pas vers un point stationnaire ;
— n'est pas strictement descendante.

Nous pouvons illustrer le deuxième phénomène avec la fonction de Rosenbrock (1.10). Regardons le comportement de l'algorithme 2 lorsqu'on utilise la direction de Newton comme direction de descente et $(\frac{1}{3}, \frac{2}{3})$ comme point de départ. Le tableau 3.1 présente les valeurs de $f(x)$ ainsi que les valeurs de la norme du gradient correspondantes. Nous pouvons observer la convergence vers un point stationnaire ($\|f(x_6)\| = 0$). Cependant, $f(x_0) < f(x_1)$ et $f(x_2) < f(x_3)$, donc nous n'avons pas une suite d'itérés où $f(x_k)$ est strictement décroissante. Nous avons simplement été « chanceux » d'obtenir une solution. Nous voulons une procédure qui nous assure d'obtenir une solution de manière systématique. L'algorithme 3 présente une telle procédure.

Itérations	$f(x)$	$\|\nabla f(x)\|$
0	31.1	1.3×10^{2}
1	430	1.3×10^{3}
2	0.23	9.5×10^{-1}
3	5.1	1.0×10^{2}
4	0.0	2.4×10^{-4}
5	0.0	6.6×10^{-6}
6	0.0	0.0×10^{0}

Tableau 3.1 – Valeur de $f(x)$ pour la fonction de Rosenbrock avec $(\frac{1}{2}, \frac{2}{3})$ comme point de départ

Algorithme 3 : Algorithme de descente avec recherche linéaire

Données : x_0 dans un voisinage de la solution, $\epsilon > 0$ et $M \in \mathbb{N}$

1 $x \leftarrow x_0$, iter$\leftarrow$0 ;
2 **tant que** *(*$\|\nabla f(x)\| < \epsilon$ ***et*** *iter* $< M$*)* **faire**
3 $\quad d \leftarrow$ direction de descente ;
4 $\quad \alpha \leftarrow$ pas de déplacement admissible ;
5 $\quad x \leftarrow x + \alpha d$;
6 $\quad$ iter $\leftarrow$ iter $+ 1$
7 **retourner** x

L'algorithme 3 est similaire à l'algorithme 2 à l'exception de la ligne 4 où un pas de déplacement α, un réel positif, a été ajouté. Le pas de déplacement permet de mettre à l'échelle notre direction de descente d et est choisi de telle sorte que $f(x_{k+1}) < f(x_k)$. Il existe de nombreux critères pour s'assurer d'obtenir un α admissible. Nous présentons deux critères répandus d'Armijo et de Wolfe :

$$f(x + \alpha d) - f(x) \leqslant \tau_0 \alpha \nabla f(x)^T d, \qquad \text{(Armijo)}$$

$$\nabla f(x + \alpha d)^T d \geqslant \tau_1 \nabla f(x)^T d. \qquad \text{(Wolfe)}$$

où $\tau_0 \in (0, \frac{1}{2})$ et $\tau_1 \in (\tau_0, 1)$. Le critère d'Armijo nous assure que $f(x + \alpha d) < f(x)$. Le critère de Wolfe nous assure qu'on s'éloigne suffisamment de notre point initial. Un

pas de déplacement α est dit admissible s'il respecte une des conditions d'Armijo ou de Wolfe. Pour trouver un pas de déplacement admissible α, on utilise un algorithme de recherche linéaire (*line search algorithm*).

La fonction $h_{x,d}(\alpha) = f(x + \alpha d)$, qui est une fonction scalaire, est souvent utilisée pour faciliter le calcul d'un $\alpha > 0$ admissible. Tenter de trouver un minimum local de la fonction $h_{x,d}$ est une recherche linéaire exacte et cette technique est coûteuse. Généralement, on se contente de trouver un point α qui satisfait aux conditions d'Armijo et/ou de Wolfe. Une stratégie populaire consiste à choisir le premier pas admissible dans $\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, ...\}$. Cette stratégie est le procédé d'Armijo.

Reprenons l'exemple de la fonction de Rosenbrock et le point de départ $(\frac{1}{3}, \frac{2}{3})$, mais en utilisant l'algorithme 3 où on effectue une recherche linéaire avec le procédé d'Armijo. À chaque itération, on s'assure que le critère d'Armijo est satisfait avec $\tau_0 = 0.01$. On consigne les résultats dans le tableau 3.2.

Itérations	$f(x)$	$\|\nabla f(x)\|$	α
0	32.3	1.3×10^{2}	0.5
1	1.96	1.8×10^{2}	1
2	8.9×10^{-3}	1.9×10^{1}	1
3	7.9×10^{-3}	4.0×10^{0}	1
4	2.1×10^{-10}	2.9×10^{-5}	1
5	4.4×10^{-18}	9.4×10^{-8}	1

Tableau 3.2 – Valeur de $f(x)$ pour la fonction de Rosenbrock avec $(\frac{1}{2}, \frac{2}{3})$ comme point de départ et recherche linéaire d'Armijo

Dans le tableau 3.2, on observe bel et bien que $f(x_{k+1}) < f(x_k)$ pour tout k. Nous observons également que le pas α n'est pas toujours un. Dans le cadre de l'algorithme 2 tous les α auraient été égaux à un. Il est possible de démontrer que, si la direction de Newton est utilisée, l'algorithme 3 converge vers une solution x^* et que dans un voisinage

de la solution le pas de déplacement α sera unitaire [Dus15].

Dans la Section 4.1 nous verrons comment la recherche linéaire peut être utilisée pour globaliser les méthodes d'ordre supérieur à l'aide d'un nouveau critère d'admissibilité différent de ceux d'Armijo et de Wolfe.

3.2 Régions de confiance

Les stratégies de région de confiance se réduisent également à la direction de Newton au voisinage de la solution. Nous utiliserons la notation TR (*trust-region methods*) pour faire référence aux stratégies de région de confiance. Il existe différentes variantes de région de confiance. Nous commençons par présenter l'idée générale et le schéma global que l'on peut retrouver dans [CGT00].

L'idée des stratégies de région de confiance consiste à approximer notre fonction objectif f par une autre fonction m_k, appelée modèle, à chaque itération. Le modèle est une fonction avec des propriétés connues dont le minimum local ou global peut être facilement calculé dans la région de confiance. On veut s'assurer que le modèle est une bonne approximation de f dans une certaine région (la région de confiance) de taille Δ_k. Si tel est le cas, se déplacer vers le minimum du modèle m_k fait en sorte que x_k se rapproche d'un minimum local de f. Si le modèle n'est pas une bonne approximation de f dans la région de confiance, l'itéré courant ne change pas ($x_{k+1} = x_k$) et la taille de la région de confiance est réduite jusqu'à ce que le modèle soit une bonne approximation. L'algorithme 4 présente une écriture plus formelle du schéma de base de région de confiance.

Regardons un peu plus en détail chacune des étapes importantes de l'algorithme 4. Nous

Algorithme 4 : Algorithme de base de région de confiance

Données : x_0 point de départ, $\epsilon > 0$, $\Delta_0 > 0$ un rayon de région de confiance initial ;

Des constantes $\eta_1, \eta_2, \gamma_1, \gamma_2$ qui satisfont $0 < \eta_1 \leqslant \eta_2 < 1$ et $0 < \gamma_1 \leqslant \gamma_2 < 1$.

$x \leftarrow x_0$, $k \leftarrow 0$;

tant que $(\|\nabla f(x)\| < \epsilon)$ **faire**

- Définir m_k un modèle qui approxime f ;
- Calculer d_k une direction qui réduit m_k dans la région de confiance Δ_k ;
- $\rho_k \leftarrow \frac{\Delta f}{\Delta m} := \frac{f(x_k) - f(x_k + d_k)}{m_k(x_k) - m_k(x_k + d_k)}$;
- **si** $\rho_k \geqslant \eta_1$ **alors**
  - $x_{k+1} = x_k + d_k$ (itération fructueuse)
  - **si** $\rho_k \geqslant \eta_2$ **alors**
    - $\Delta_{k+1} \in [\Delta_k, \infty]$ (itération très fructueuse)
- **sinon si** $\rho_k \in [\eta_1, \eta_2)$ **alors**
  - $\Delta_{k+1} \in [\gamma_2 \Delta_k, \Delta_k]$ (itération infructueuse)
- **sinon si** $\rho_k < \eta_1$ **alors**
  - $\Delta_{k+1} \in [\gamma_1 \Delta_k, \gamma_2 \Delta_k]$ (itération très infructueuse)
- $k \leftarrow k + 1$;

retourner x

utilisons l'approximation quadratique $q_x(d)$ (série de Taylor limité d'ordre deux de la fonction f) comme modèle m_k. Il est possible d'utiliser d'autres modèles, le choix de q_x est le plus répandu. Par la suite, pour calculer d_k, une direction qui réduit q_k dans la région de confiance Δ_k, il faut résoudre

$$\begin{aligned} \min_{d} \quad & q_x(d) \\ \text{s.à} \quad & \|d\| \leqslant \Delta_k. \end{aligned} \tag{3.1}$$

Résoudre (3.1) est la partie la plus exigeante en termes de calculs dans l'algorithme 4. Il s'agit du sous-problème de région de confiance. Nous regarderons l'approche de Moré-Sorensen pour résoudre (3.1) dans la section 3.2.1.

Le paramètre ρ_k (ligne 5) évalue la qualité du modèle dans la région de confiance Δ_k. Si m_k est une bonne approximation de f alors les quantités $\Delta f = f(x_k) - f(x_k + d_k)$ et $\Delta m = q_k(0) - q_k(d_k)$ sont similaires et $\rho_k \approx 1$. Autrement dit, une réduction du modèle se traduit par une réduction proportionnelle de la fonction objectif. Si m_k n'est pas une bonne approximation, alors ρ_k est différent de 1, ρ_k peut même être négatif ou plus grand que 1.

Les paramètres η_1 et η_2 sont utilisés pour qualifier la qualité de l'approximation (lignes 6 à 15). Si $\rho_k \geqslant \eta_1$, alors q_x est une bonne approximation et x_k se déplace dans la direction d_k, le minimum de q_x dans la région de confiance (ligne 4). Si q_x est une très bonne approximation ($\rho \geqslant \eta_2$), on se permet d'augmenter la taille de la région de confiance (ligne 8). Les lignes 10 à 15 traitent les cas où q_x est une mauvaise approximation. Dans un tel cas, $x_{k+1} = x_k$ et la taille de la région de confiance est réduite.

Le but de la recherche linéaire est d'obtenir un pas de déplacement $\alpha = 1$ afin d'utiliser la direction de Newton. Le schéma de région de confiance se réduit également à la direction de Newton. En effet, la direction de Newton est le minimum local de $q_x(d)$ lorsque la

hessienne de f est définie positive. Dans un voisinage d'une solution x^*, la taille de la région de confiance Δ est assez grande et la contrainte $\|d\| \leqslant \Delta$ n'est pas active. Donc, la solution de (3.1) devient la direction de Newton.

Avant de présenter le théorème de convergence de l'algorithme 4, nous présentons le pas de Cauchy.

Définition 3.1 *Le pas de Cauchy d_{Ca} est tel que*

$$d_{Ca} = \arg \min_{\substack{t \geqslant 0 \\ \| -t\nabla f(x_k)\| \leqslant \Delta_k}} q_x(t\nabla f(x_k)). \tag{3.2}$$

Le pas de Cauchy est la réduction du modèle quadratique de la direction de la pente la plus forte, i.e. $-\nabla f(x_k)$, dans la région de confiance Δ_k. Le pas de Cauchy est important pour prouver la convergence des algorithmes de région de confiance. Le théorème 3.2 présente une borne inférieure de la décroissance du modèle quadratique lorsqu'on utilise le pas de Cauchy (disponible dans [CGT00]).

Théorème 3.2 *Soit $f \in \mathcal{C}^2$ et $q_x(d)$ définie par (1.1). Posons*

$$\beta_k = 1 + \max_{\|x_k\| \leqslant \Delta_k} \|\nabla^2 f(x_k)\|.$$

La décroissance du modèle quadratique dans la direction de Cauchy est bornée inférieurement par :

$$q_x(0) - q_x(d_{Ca}) \geqslant \tfrac{1}{2}\|\nabla f(x_k)\| \min\{\frac{\|\nabla f(x_k)\|}{\beta_k}, \Delta_k\}. \tag{3.3}$$

La direction de Cauchy d_{Ca} est utile pour définir les directions qui assurent la convergence de l'algorithme 4. En effet, pour que l'algorithme 4 converge vers un point stationnaire il suffit que la direction choisie produise une fraction (κ_{mdc}) de la décroissance du modèle lorsque le pas de Cauchy est utilisé, c'est-à-dire que d doit satisfaire :

$$q_x(0) - q_x(d) \geqslant \kappa_{mdc}\tfrac{1}{2}\|\nabla f(x_k)\| \min\{\frac{\|\nabla f(x_k)\|}{\beta_k}, \Delta_k\}, \text{ où } \kappa_{mdc} \in (0,1). \tag{3.4}$$

On note que si d est solution de (3.1) alors d satisfait (3.4). Le théorème 3.3 résume les hypothèses et conditions pour que l'algorithme 4 converge vers un point stationnaire.

Théorème 3.3 *Supposons que f soit bornée inférieurement et que $\|\nabla^2 f(x_k)\| \leqslant \kappa$ pour tout k. Supposons qu'on utilise l'approximation quadratique, i.e $m_k(d) = q_x(d)$. Finalement, supposons que la direction d_k soit telle que (3.4) est satisfaite et $\|d_k\| \leqslant \gamma\Delta_k$ où $\gamma > 1$. Alors, la séquence $\{x_k\}$ produite par l'algorithme 4 est telle que*

$$\lim_{k\to\infty} \|\nabla f(x_k)\| = 0.$$

Autrement dit, la suite $\{x_k\}$ produite par l'algorithme 4 converge vers un point stationnaire.

La preuve est disponible dans [CGT00]. Il est possible de simplifier l'écriture de l'algorithme 4 et de réduire la quantité de calculs nécessaires. Nous présentons l'algorithme 5, une manière équivalente et plus simple d'écrire l'algorithme 4 proposé dans [Dus19].

Faisons quelques observations sur l'algorithme 5. Tout d'abord, on remarque que tant qu'une itération n'est pas fructueuse le modèle quadratique n'est pas modifié. En effet, dans l'algorithme 4 tous les paramètres relatifs à $q_x(d)$ sont calculés à nouveau à chaque itération infructueuse. Ces calculs redondants sont épargnés dans l'algorithme 5. L'algorithme 5 permet également d'introduire facilement les méthodes ARC, que nous verrons dans la Section 3.3. Les autres changements visent à rendre l'algorithme plus lisible.

Dans la section 3.2.1 nous présenterons l'approche de Moré-Sorensen qui utilise des décompositions de matrices pour résoudre le sous-problème de région de confiance. Il existe également l'approche de Steihaug-Toint pour résoudre le sous-problème de région de

Algorithme 5 : Algorithme ARCTR

Données : x_0 point de départ, $\epsilon > 0$, Δ_0 un rayon de région de confiance initial ;
Des constantes $\eta_1, \eta_2, \gamma_1, \gamma_2$ qui satisfont
$0 < \eta_1 \leqslant \eta_2 < 1$ et $0 < \gamma_1 < 1 < \gamma_2$.

$x \leftarrow x_0$, k$\leftarrow$0 ;
tant que $\|\nabla f(x)\| < \epsilon$ **faire**
  Définir m_k un modèle qui approxime f ;
  répéter
    Calculer d_k une direction qui réduit m_k dans la région de confiance Δ_k ;
    $\rho \leftarrow \frac{\Delta f}{\Delta m} = \frac{f(x_k)-f(x_k+d_k)}{m_k(x_k)-m_k(x_k+d_k)}$;
    si $\rho < \eta_1$ **alors** $\Delta = \gamma_1 \Delta$ (itération infructueuse);
    sinon
      $x \leftarrow x + d$ (itération fructueuse) ;
      si $\rho > \eta_2$ **alors** $\Delta = \gamma_2 \Delta$ (itération très fructueuse);
  jusqu'à *(on a une itération (très) fructueuse)*;
  $k \leftarrow k + 1$;
retourner x

confiance. Cette approche utilise des méthodes itératives, comme le gradient conjugué, pour résoudre des systèmes linéaires. Les méthodes itératives débordent du cadre de ce mémoire.

La terminologie « algorithme de région de confiance » fait donc référence à l'algorithme 5 lorsque q_x est utilisé comme modèle et que le paramètre Δ est utilisé pour la taille de la région de confiance.

3.2.1 Décompositions de matrices

L'approche de Moré-Sorensen pour résoudre (3.1) consiste à transformer la hessienne $\nabla^2 f(x)$ et résoudre un système linéaire modifié. Le théorème 3.4 et le corollaire 1, donnés dans [CGT00], présentent la matrice modifiée et le nouveau système linéaire.

Théorème 3.4 *Tout minimum global de $q_x(d)$ sujet à $\|d\| = \Delta$ satisfait l'équation :*

$$H(\lambda)d(\lambda) = -\nabla f(x), \tag{3.5}$$

où $H(\lambda) := \nabla^2 f(x) + \lambda I$ est semi-définie positive. Si $H(\lambda)$ est définie positive, alors $d(\lambda)$ est unique.

Il est important de remarquer que dans (3.1), la contrainte se présente avec une inégalité, tandis que la solution de (3.5) satisfait une contrainte d'égalité. Le corollaire 1 traite le cas sujet à une contrainte d'inégalité.

Corollaire 1 *Tout minimum global de $q_x(d)$ sujet à $\|d\| \leqslant \Delta$ satisfait l'équation :*

$$H(\lambda)d(\lambda) = -\nabla f(x), \tag{3.6}$$

où $H(\lambda)$ est semi-définie positive, $\lambda \geqslant 0$ et $\lambda\,(\|d(\lambda)\| - \Delta) = 0$. Si $H(\lambda)$ est définie positive, alors $d(\lambda)$ est unique.

Nous notons que si $\lambda = 0$, cela signifie que la solution de (3.6) est la direction de Newton. L'approche de Moré-Sorensen consiste donc à trouver λ et $d(\lambda)$ des solutions de (3.1) qui satisfont aux conditions présentées dans le corollaire 1.

Pour l'implémentation numérique, nous utilisons un schéma proposé dans [Dus19] pour résoudre (3.1) et trouver λ qui satisfait aux conditions du corollaire 1. Ce schéma consiste à factoriser la hessienne de telle sorte que $\nabla^2 f(x) = A\Lambda A^T$ où A est facilement inversible (triangulaire ou orthogonale) et Λ est une matrice diagonale.

Les factorisations LDL^T et Bounded-Bunch-Kaufman permettent d'obtenir une telle factorisation où $A = S^{-1}P^T LQ$ où $Q\Lambda Q^T$ est la décomposition spectrale de D et les matrices S et P sont des matrices de mise à l'échelle et de permutation respectivement, introduites à la section 1.3. Si D est diagonale alors Q est la matrice identité, mais on rappelle que

dans la décomposition de Bunch-Kaufman, D est diagonale par bloc 1×1 ou 2×2. On note que A est une matrice inversible.

Le système à résoudre est le suivant :

$$(A\Lambda A^T + \lambda I)d = -\nabla f(x).$$

En posant $\tilde{d} = A^T d$ et $A\tilde{g} = \nabla f(x)$ le système devient

$$(\Lambda + \lambda I)\tilde{d} = -\tilde{g}. \tag{3.7}$$

Le système (3.7) est facile à résoudre, car le terme de gauche est une matrice diagonale. Donc λ est choisi de tel sorte que $(\Lambda + \lambda I)$ soit semi-définie positive. Un choix naturel est de prendre $\lambda = \max(0, -\lambda_1)$ où λ_1 est la plus petite valeur propre de Λ. On cherche donc un λ tel que

$$\begin{aligned} (\Lambda + \lambda I)\, \tilde{d}(\lambda) &= -\tilde{g}, \\ \lambda &\geqslant 0, \\ (\Lambda + \lambda I) &\succeq 0. \end{aligned}$$

Les détails sont présentés dans [Dus19]. Il est important de mentionner que les « cas difficiles » de région de confiance sont également traités par ce schéma.

Finalement, à partir de $\tilde{d}$ il est possible de retrouver d, $d = A^{-t}\tilde{d}$, étant donné que la matrice A a été construite pour être facilement inversible.

3.2.2 Exemple numérique

Pour terminer la présentation des région de confiance, voici un exemple numérique pour illustrer l'approche de Moré-Sorensen dans le schéma de région de confiance lorsque

$\nabla^2 f(x)$ est une matrice dense. La fonction de Rosenbrock (1.10) est encore utilisée avec comme point de départ $x_0 = (\frac{1}{3}, \frac{2}{3})$.

Le tableau 3.3 présente différentes données d'un algorithme de région de confiance qui utilise l'approche de Moré-Sorenson pour résoudre le sous-problème de région de confiance. La colonne « status » nous indique si l'itéré a été modifié, i.e. $x_{k+1} = x_k + d_k$ ou si $x_{k+1} = x_k$. Donc, si l'itération est infructueuse la valeur correspondante dans la colonne $f(x)$ indique quelle aurait été la valeur si on avait choisi $x_{k+1} = x_k + d$ et non $x_{k+1} = x_k$. Les valeurs de $f(x)$ sont donc monotones décroissantes pour les itérations (très) fructueuses. On observe la convergence quadratique de la méthode de Newton à l'aide de la norme du gradient, c'est-à-dire qu'à partir de la douzième itération $\|\nabla f(x_{k+1})\| \leqslant \|\nabla f(x_k)\|^2$. La valeur de λ pour ces itérations est nulle.

Itérations	$f(x)$	$\|\nabla f(x)\|$	λ	Δ	status
0	3.1×10^{1}	1.3×10^{2}	0.0×10^{0}	1.0×10^{1}	fructueuse
1	3.1×10^{1}	1.3×10^{2}	1.4×10^{2}	1.0×10^{1}	infructueuse
2	3.1×10^{1}	1.3×10^{2}	2.1×10^{2}	1.0×10^{0}	infructueuse
3	1.3×10^{1}	7.1×10^{1}	1.8×10^{3}	5.0×10^{1}	Très fructueuse
4	3.4×10^{-1}	2.1×10^{1}	1.4×10^{2}	5.0×10^{1}	fructueuse
5	4.9×10^{-3}	1.3×10^{1}	0.0×10^{0}	2.5×10^{0}	Très fructueuse
6	2.3×10^{-3}	1.9×10^{0}	0.0×10^{0}	2.5×10^{0}	fructueuse
7	6.4×10^{-8}	4.8×10^{-4}	0.0×10^{0}	1.3×10^{1}	Très fructueuse
8	4.1×10^{-13}	2.6×10^{-5}	0.0×10^{0}	6.3×10^{1}	Très fructueuse
9	2.0×10^{-27}	1.3×10^{-13}	0.0×10^{0}	3.1×10^{2}	Très fructueuse

Tableau 3.3 – Algorithme de région de confiance qui utilise l'approche de Moré-Sorensen pour résoudre le sous-problème.

3.3 ARC

La stratégie ARC (*Adaptive Regularization using Cubics*) a été introduite dans [CGT11a, CGT11b]. Il est possible de voir la stratégie ARC comme une variante des région de confiance classique. En effet, il est possible de l'intégrer dans l'algorithme 5. Cependant il y a deux différences majeures :

— le choix du modèle pour approximer f,

— le rôle du paramètre Δ est pris en charge par un nouveau paramètre α.

Commençons par regarder le nouveau modèle qui approxime la fonction objectif. Pour les région de confiance $m_x = q_x$, tandis que pour la stratégie ARC $m_x = c_x$:

$$c_x^\alpha(d) = f(x) + \nabla f(x)^T d + \tfrac{1}{2} d^T \nabla^2 f(x) d + \tfrac{1}{3\alpha}\|d\|^3. \tag{3.8}$$

On remarque que $c_x^\alpha(d) = q_x(d) + \frac{1}{3\alpha}\|d\|^3$. Le terme $\frac{1}{3\alpha}\|d\|^3$ est une régularisation cubique et le paramètre α est strictement positif. Pour les région de confiance, la direction d (ligne 5 de l'algorithme 5) est la solution de (3.1). Lorsque la région de confiance Δ est grande, la solution de ce problème est la direction de Newton, soit le minimum de q_x lorsque cette fonction est convexe. Plus la région de confiance est petite, plus il est probable que la solution soit sur la frontière de la région de confiance, donc dans un voisinage de l'itéré courant. Donc un petit Δ signifie un plus grand impact de la région de confiance.

La terminologie « algorithme ARC » est utilisée lorsque l'algorithme 5 est utilisé avec le modèle c_x^α et le paramètre α à la place de Δ. Si on veut utiliser ARC, le nouveau sous-problème à résoudre est donc

$$\min_d \quad c_x^\alpha(d). \tag{3.9}$$

Tout comme pour les régions de confiance, si α est grand le terme de régularisation cubique est petit et donc le minimum de c_x est atteint dans une direction similaire à celle

de Newton. En effet, α est strictement positif, donc le terme de régularisation cubique est toujours non nul et $c_x^\alpha \neq q_x$. Bien qu'asymptotiquement la direction issue de ARC s'apparente à la direction de Newton, elles ne sont jamais égales. Alternativement, un petit α signifie que le terme de régularisation cubique est plus important, il s'agit d'une pénalisation. Une grande pénalisation signifie que la solution de (3.9) est dans un petit voisinage de l'itéré courant.

Le corollaire 1 explicite les conditions que doivent respecter x^* et λ^* pour être solutions de (3.1). Les conditions suivantes caractérisent une solution de 3.9

$$\left(\nabla^2 f(x) + \lambda I\right) d = -\nabla f(x)^t, \tag{3.10}$$

$$\lambda \geq 0, \tag{3.11}$$

$$\left(\nabla^2 f(x) + \lambda I\right) \succeq 0, \tag{3.12}$$

$$\|d\| = \alpha\lambda. \tag{3.13}$$

Pour trouver λ et d qui satisfont (3.10)-(3.13) il est possible d'utiliser l'approche de Moré-Sorensen. Comme le terme de régularisation cubique ne se réduit pas à zéro, λ est donc toujours strictement positif.

Il est important de mentionner que la version traditionnelle de l'algorithme ARC, le paramètre ρ calculé à la ligne 6 de l'algorithme 5 est tel que :

$$\rho = \frac{\Delta f}{\Delta c} = \frac{f(x_k) - f(x_k + d_k)}{c_k^\alpha(0) - c_k^\alpha(x_k + d_k)}.$$

Cependant, dans notre implémentation numérique nous utilisons :

$$\rho = \frac{\Delta f}{\Delta q} = \frac{f(x_k) - f(x_k + d_k)}{q_k(x_k) - q_k(x_k + d_k)}.$$

Lorsque ρ est défini en fonction du modèle quadratique, il est plus juste d'utiliser la notation « algorithme ARCq » au lieu de « algorithme ARC ». L'algorithme ARCq est une variante de l'algorithme ARC proposée dans [Dus18]. Puisque ARC et ARCq partagent

leurs propriétés de convergence globale et de complexité algorithmique, la notation ARC est utilisée pour la suite du document.

Avant de présenter un exemple numérique, nous démontrons que l'algorithme 5 converge dans sa version ARC. Nous présentons tout d'abord un théorème sur les conditions d'optimalité.

Théorème 3.5 *Si x^* est un minimum local de f, alors les deux conditions suivantes sont équivalentes :*

1. $\nabla f(x^*)$ *et* $\nabla^2 f(x^*) \succeq 0$ *;*
2. $\forall \alpha > 0$, $d = 0$ *est un minimum global de* c_k^α

La preuve est disponible dans [Dus18]. Nous démontrons maintenant la convergence globale de l'algorithme ARC.

Théorème 3.6 *Soit $\{x_k\}$ la suite de points engendrés par l'algorithme 5 dans sa version ARC. Soit $f \in \mathcal{C}^2(\mathbb{R}^n)$ telle que $f(x_k)$ est bornée inférieurement et que $\{x_k\}$ demeure dans un compact $C \subset \mathbb{R}^n$, alors il existe une sous-suite $\{x_{k_i}\}$, extraite de $\{x_k\}$, qui converge vers un point d'accumulation x^* qui satisfait aux conditions nécessaires d'ordre deux. Plus précisément, pour toute sous-suite $\{x_{k_i}\}$, extraite de $\{x_k\}$, qui converge vers un point d'accumulation x^*, ce point d'accumulation satisfait aux conditions nécessaires d'ordre deux.*

Démonstration. L'hypothèse de compacité assure que toute sous-suite possède un point d'accumulation, donc $\{x_{k_i}\}$ admet un point d'accumulation $\bar{x}$. Selon Fletcher [Fle87], l'analyse se divise en deux cas :

1. la suite $\{\alpha_k\}$ est strictement positive donc on peut extraire une sous-suite telle que $\alpha_{k_i} \geqslant \bar{\alpha} > 0$ et $r_{k_i} \geqslant \eta_1$ pour tout i,

2. la suite $\{\alpha_k\}$ converge vers zéro.

Nous traitons seulement du premier cas, le second cas est présenter dans [Dus18]. Donc nous supposons que $\alpha_{k_i} \geqslant \bar{\alpha} > 0$ pour le reste des développements.

Montrons que tout point d'accumulation $\bar{x}$ d'une sous-suite $\{x_{k_i}\}$ satisfait aux conditions nécessaires d'optimalité du théorème 3.5. Les itérations fructueuses et très fructueuses assurent la décroissance $f(x_k)$ tandis que les itérations infructueuses ne modifient rien, i.e. $x_{k+1} = x_k$. Donc, $f(x_k)$ est bornée inférieurement et strictement décroissante, il s'agit d'une suite de Cauchy qui converge vers une valeur finie et $\Delta f(x_k)$ converge vers zéro. Si $\Delta f(x_k)$ converge vers zéro, on a également que $\Delta q(d_k)$ converge vers zéro.

Maintenant considérons les quantités suivantes :

— $\bar{d}$ une solution optimale de $\min c_{\bar{x}}^{\bar{\alpha}}$;

— $\hat{x} = \bar{x} + \bar{d}$;

— $\hat{d_{k_i}} = \hat{x} - x_{k_i}$.

En posant d_{ARC_k} le minimum de $c_{x_k}^{\alpha_k}$ on obtient :

$$c_{x_{k_i}}^{\bar{\alpha}}(\hat{d_{k_i}}) \geqslant c_{x_{k_i}}^{\alpha_{k_i}}(\hat{d_{k_i}}) \geqslant c_{x_{k_i}}^{\alpha_{k_i}}(d_{\mathrm{ARC}_{k_i}}) = f(x_{k_i}) - \Delta c_{x_{k_i}}^{\alpha_{k_i}}(d_{\mathrm{ARC}_{k_i}}). \tag{3.14}$$

Montrons que $\Delta m_{x_{k_i}}^{\alpha_{k_i}}(d_{ARC_{k_i}})$ converge à zéro. On a :

$$0 \geqslant -\Delta c_{x_{k_i}}^{\alpha_{k_i}}(d_{ARC_{k_i}}) = -\Delta q_{x_{k_i}} + \tfrac{1}{3\alpha_k}\|d_{\mathrm{ARC}_{k_i}}\|^3 \geqslant -\Delta q_{x_{k_i}}.$$

Étant donné que $\Delta q(d_k) \to 0$, selon le théorème des gendarmes nous avons que $\Delta c_{x_{k_i}}^{\alpha_{k_i}}(d_{\mathrm{ARC}_{k_i}})$ tend vers zéro également. Donc en prenant les limites des termes de gauche et de droite de (3.14) on obtient :

$$c_{\bar{x}}^{\bar{\alpha}}(\bar{d}) \geqslant f(\bar{x}) = c_{\bar{x}}^{\bar{\alpha}}(0).$$

Puisque 0 est un minimum global de $c_{\bar{x}}^{\bar{\alpha}}$, selon le théorème 3.5, $\bar{x}$ satisfait aux conditions nécessaires d'ordre deux. ■

3.3.1 Exemple numérique

Pour conclure cette section sur l'algorithme ARC nous présentons un exemple numérique. Nous appliquons la méthode ARC sur la fonction de Rosenbrock avec $x_0 = (\frac{1}{3}, \frac{2}{3})$, les résultats sont consignés dans le tableau 3.4. Nous pouvons observer que le terme λ est strictement positif pour toutes les itérations, mais réduit au fil des itérations (très) fructueuses. Le terme α quant à lui augmente au fil des itérations. Le terme de régularisation cubique est d'importance réduite dans les dernières itérations d'où une convergence quadratique qui s'apparente à la convergence de la méthode de Newton. Notons que si une itération est infructueuse alors le $f(x)$ correspondant est le $f(x+d)$ qui a été rejeté.

Itérations	$f(x)$	$\|\nabla f(x)\|$	λ	status	α
0	3.1×10^{1}	1.3×10^{2}	0.0×10^{0}	fructueuse	1.0×10^{1}
1	3.1×10^{1}	1.3×10^{2}	1.3×10^{2}	infructueuse	1.0×10^{1}
2	3.1×10^{1}	1.3×10^{2}	1.3×10^{2}	infructueuse	1.0×10^{0}
3	3.1×10^{1}	1.3×10^{2}	1.4×10^{2}	infructueuse	1.0×10^{-1}
4	3.1×10^{1}	1.3×10^{2}	1.8×10^{2}	infructueuse	1.0×10^{-2}
5	1.2×10^{1}	1.2×10^{2}	3.9×10^{2}	fructueuse	1.0×10^{-3}
6	1.2×10^{0}	3.2×10^{1}	1.6×10^{2}	très fructueuse	5.0×10^{-3}
7	7.2×10^{-2}	2.0×10^{0}	1.9×10^{1}	très fructueuse	2.5×10^{-2}
8	4.5×10^{-2}	5.5×10^{-1}	2.6×10^{0}	très fructueuse	1.3×10^{-1}
9	2.2×10^{-2}	2.0×10^{0}	1.0×10^{0}	très fructueuse	6.3×10^{-1}
10	7.4×10^{-3}	8.6×10^{-1}	1.5×10^{-1}	très fructueuse	3.1×10^{0}
11	1.6×10^{-3}	9.4×10^{-1}	3.1×10^{-2}	très fructueuse	1.6×10^{1}
12	1.3×10^{-4}	1.4×10^{-1}	2.7×10^{-3}	très fructueuse	7.8×10^{1}
13	1.6×10^{-6}	3.6×10^{-2}	2.6×10^{-4}	très fructueuse	3.9×10^{2}
14	2.7×10^{-10}	2.2×10^{-4}	4.4×10^{-6}	très fructueuse	2.0×10^{3}
15	8.9×10^{-18}	8.8×10^{-8}	1.8×10^{-8}	très fructueuse	9.8×10^{3}

Tableau 3.4 – Algorithme ARC appliqué à la fonction de Rosenbrock.

3.4 Comparaisons d'algorithmes

Pour comparer le comportement de différentes méthodes sur une collection de problèmes nous utilisons les profils de performance [DM02]. Les profils de performance sont des graphiques qui permettent de comparer la performance de différents algorithmes selon différents critères, par exemple le temps de calcul nécessaire pour atteindre un point stationnaire, le nombre d'itérations ou le nombre d'évaluations de f.

Chaque algorithme est associé à une courbe sur le profil. Les différentes courbes donnent différentes informations sur les algorithmes.

L'axe vertical représente la quantité de problèmes résolus en pourcentage. Un algorithme dont la courbe se stabilise en 1.00 résout donc 100% des problèmes de la collection. Un algorithme qui résout un haut pourcentage de problème est considéré comme robuste. Par exemple, la figure 3.1 présente un profil de performance qui compare un algorithme de région de confiance et un algorithm ARC sur une sélection de 150 problèmes tirés de la collection `CUTEst` [GOT15]. La dimension des problèmes varie entre 2 et 10^5. Toutes les matrices sont encodées de manière creuse et la double précision (`Float64`) est utilisée. On peut voir que l'algorithme ARC résout un peu moins de problèmes que TR. Les deux algorithmes résolvent environ 90% des problèmes.

L'axe horizontal représente la « vitesse » des algorithmes selon le critère choisi. L'axe des ordonnées indique le pourcentage de problèmes qu'un algorithme a résolu plus rapidement que les autres. Il s'agit de la vitesse « absolue » des algorithmes. Par exemple, à la figure 3.1 le critère de performance utilisé est le nombre d'itération. Donc, l'algorithme de région de confiance prend moins d'itérations pour 70% des problèmes. L'algorithme ARC est le plus rapide pour 40% des problèmes. Il est possible que la somme des pourcentages soit supérieur à 100%. En effet, il est possible que deux algorithmes nécessitent le même nombre d'itération pour résoudre un même problème. Pour une mesure de performance

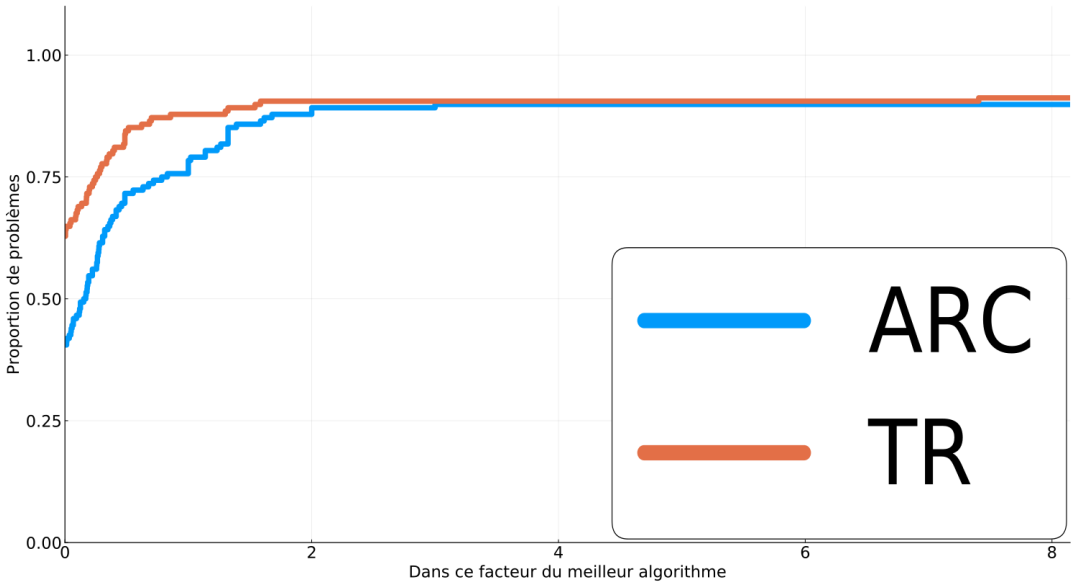

Figure 3.1 – Comparaison d'un algorithme de région de confiance avec un algorithme ARC en fonction du nombre d'itérations

comme le temps de calcul, il est plus rare d'avoir de nombreux algorithmes qui nécessitent le même temps de calcul pour atteindre une solution, donc les valeurs de chaque courbe à l'axe des ordonnées somme généralement à 100%.

L'axe des ordonnées indique la vitesse absolue des algorithmes. Les autres valeurs de l'axe horizontal indiquent la vitesse relative des algorithmes par rapport à un certain facteur logarithmique (nous utilisons un facteur logarithmique 2 dans ce document). Donc, si deux courbes rejoignent en 1.0 sur l'axe horizontal, cela signifie que dans un facteur logarithmique d'ordre 2^1 les deux algorithmes qui correspondent aux courbes sont équivalent en termes de rapidité. Par exemple, à la figure 3.1 les courbes se rejoignent en 2.0 sur l'axe horizontal, donc à plus ou moins 2^2 itérations les algorithmes de région de confiance et de régularisation cubique sont équivalents.

La figure 3.2 présente les mêmes résultats, mais avec le temps de calcul comme mesure de

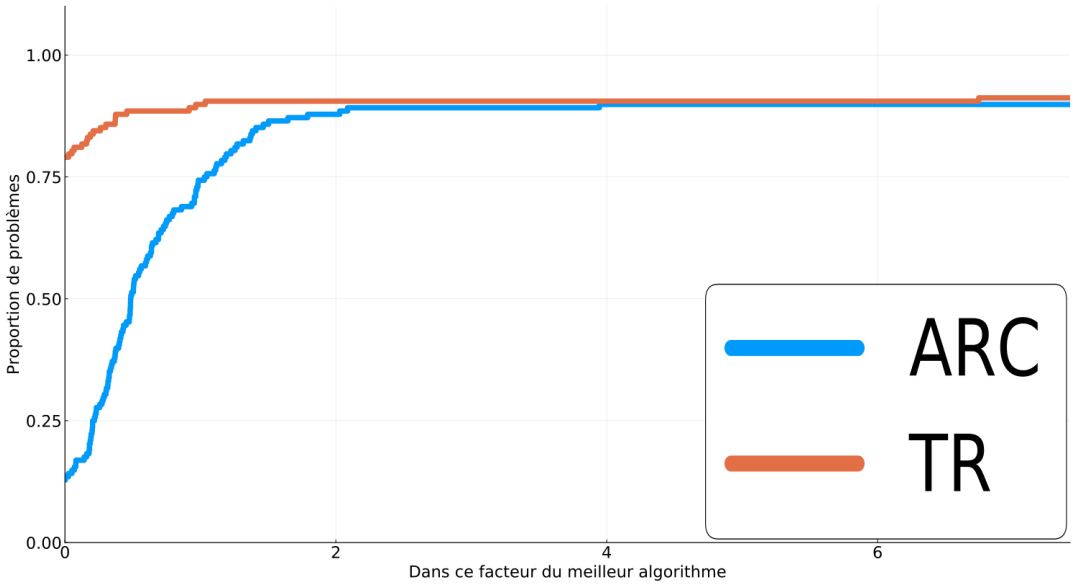

Figure 3.2 – Comparaison d'un algorithme de région de confiance avec un algorithme ARC en fonction du temps de calcul

performance. On remarque que nombre total de problèmes résolus pour chaque algorithme est le même sur les figures 3.2 et 3.1. Cependant, `TR` prend moins de temps de calcul pour 80% des problèmes.

Les profils de performance sont donc très utiles pour comparer différents algorithmes selon différentes mesures. Ils offrent de l'information rapide sur les tendances générales de différents algorithmes sur une collection de problèmes. Malgré tout, il est possible que l'information soit mal représentée. S'il y a trois algorithmes A, B et C représentés dans le profil de performance, l'algorithme A peut paraître plus « rapide » que B et C, mais si on enlève l'algorithme C, l'algorithme B peut paraître plus rapide que l'algorithme A. Ce phénomène est documenté dans [GS16]. Il faut donc faire preuve de prudence lors de l'analyse de profils de performance.

CHAPITRE 4

Globalisation des méthodes d'ordre supérieur

Dans le chapitre 2, les méthodes d'ordre supérieur (Chebyshev, (Super)Halley et Shamanskii) ont été présentées dans un contexte de convergence locale, c'est-à-dire lorsque x_0 est dans un voisinage d'une solution x^*. Dans le chapitre 3, différentes stratégies pour assurer la convergence vers x^* avec un x_0 quelconque ont été présentées. Les stratégies de recherche linéaire et de région de confiance avaient comme point commun de se ramener à la méthode de Newton proche d'une solution x^*. La stratégie ARC se réduit asymptotiquement à une approximation de la direction de Newton.

Dans ce chapitre, nous présentons de nouvelles idées qui utilisent des directions d'ordre supérieur dans des schémas de convergence globale. Nous présentons une globalisation par recherche linéaire et ensuite nous introduisons de nouvelles idées de globalisation des méthodes d'ordre supérieur par région de confiance. Les notations suivantes seront utilisées dans ce chapitre :

— d_{TR} : la direction naturellement produite par l'algorithme de région de confiance

(la solution de (3.1)),

— d_{ARC} : la direction naturellement par l'algorithme ARC, i.e. la solution de (3.9),

— d_{N} : la direction de Newton,

— d_{HO} : une direction d'ordre supérieur parmi $d_{\mathrm{C}}, d_{\mathrm{SH}}, d_{\mathrm{H}}$ ou d_{S}.

4.1 Globalisation par recherche linéaire

Il existe peu de travaux concernant la globalisation des méthodes d'ordre supérieur. La méthode de globalisation par recherche linéaire proposée par Lampariello et Sciandrone dans [LS01] est, au meilleur de nos connaissances, la plus répandue. Cette méthode ne concerne que la globalisation de la méthode de Shamanskii.

L'algorithme 3 présente l'idée générale des algorithmes basés sur une recherche linéaire. Nous avons vu qu'un pas de déplacement qui satisfait aux conditions d'Armijo et de Wolfe assure une décroissance monotone de $\{f(x_k)\}$ et la convergence vers un point stationnaire lorsque la direction de Newton est utilisée. Cependant, ces critères ne sont pas suffisants pour garantir la convergence si on utilise la direction de Shamanskii. En effet, pour calculer la direction de Shamanskii la résolution de deux systèmes linéaires avec la même matrice est nécessaire, ce qui invalide les preuves de convergence globale qui utilisent les critères d'Armijo et de Wolfe.

Dans [LS01] un nouveau critère d'admissibilité du pas de déplacement est introduit. Soit θ le pas de déplacement et $\gamma \in (0, \frac{1}{2})$, alors le critère d'admissibilité de Lampariello et Sciandrone est :

$$f(x + \theta d) \leqslant f(x) - \gamma \alpha^3 \|d\|^3. \tag{4.1}$$

En plus de proposer un nouveau critère d'admissibilité, les auteurs proposent une va-

riante de l'algorithme 3. La principale différence est la manière avec laquelle la direction de Shamanskii est calculée. La variante proposée permet de réutiliser la factorisation de la hessienne plus d'une fois, tant que cette factorisation est appropriée. Donc, contrairement à la méthode de Shamanskii présentée à la section 2.3 où deux systèmes linéaires sont résolus avec la même décomposition de la hessienne, l'algorithme de Lampariello et Sciandrone rend possible la résolution de p systèmes linéaires avec la même décomposition. L'algorithme 6 présente leur idée de manière formelle. Dans cet algorithme la notation $\hat{H}$ est utilisée pour la décomposition LDL^T de $\nabla^2 f(x)$. Si $\nabla^2 f(x)$ n'est pas définie positive, alors $\hat{H}$ correspond à une décomposition LDL^T modifiée qui permet d'assurer que $\hat{H}$ est définie positive.

Algorithme 6 : Algorithme de descente adaptatif

Données : $x_0 \in \mathbb{R}^n$, $p \geqslant 1$ un entier, $\epsilon > 0$, $c_\alpha > 0$

$x_k \leftarrow x_0$, $k \leftarrow 0$;
$k \leftarrow 0$, $i \leftarrow 0$, $u \leftarrow faux$;
tant que *(*$\|\nabla f(x)\| \geqslant \epsilon$*)* **faire**
  si $k = ip$ *ou* u **alors**
    Calculer $\nabla^2 f(x_k)$ et sa décomposition $LDL^T = \hat{H}(x_k)$. **si** $\hat{H}$ *est « trop différent » de* $\nabla^2 f(x_k)$ **alors** $u \leftarrow vrai$;
    si $k = ip$ **alors** $i = i + 1$;
  sinon
    $\hat{H}(x_k) \leftarrow \hat{H}(x_{k-1})$;
  Résoudre $\hat{H}(x_k)d_k = -\nabla f(x_k)$ et trouver α_k qui satisfait (4.1) ;
  si *(*$\alpha_k > c_\alpha$*) ou (la direction produit une décroissance suffisante)* **alors**
    $u \leftarrow faux$ et $x_{k+1} = x_k + \alpha_k d_k$
  sinon
    $u \leftarrow vrai$
  $k \leftarrow k + 1$;
retourner x_k

Les variables k et i sont des compteurs. Le compteur k représente le nombre de directions d_k calculées. Le compteur i sert à déterminer à partir de quelle itération la hessienne est

calculée. Le paramètre u indique si la décomposition est adéquate pour la hessienne au point courant ou non. Dans une situation optimale, l'algorithme calcule une nouvelle décomposition toutes les p itérations. Donc $p = 1$ correspond à la méthode de Newton et $p = 2$ correspond à la méthode de Shamanskii telle que présentée à la section 2.3. Il est toutefois possible que la direction d_k calculée à la ligne 7, ne sois pas une direction qui réduit suffisamment la fonction objectif (ligne 11), dans un tel cas, la hessienne est calculée à nouveau ainsi qu'une nouvelle décomposition (ligne 4).

Malgré le fait que l'algorithme stipule que le pas de déplacement doit satisfaire (4.1) en théorie, en pratique un pas de déplacement qui satisfait la condition d'Armijo produit des résultats numériques sembables. Le critère (4.1) semble donc avoir été proposé pour les preuves de convergence que l'on trouve dans [LS01].

L'algorithme 6 est globalement convergent et dans le pire des cas il se réduit asymptotiquement à la méthode de Newton. Autrement dit la hessienne est calculée explicitement à chaque itérations et la convergence asymptotique est quadratique. Dans le cas idéal, la matrice hessienne est évaluée à chaque p itérations et la convergence asymptotique est d'ordre $p + 1$.

Cet algorithme a deux limitations importantes. Premièrement, il ne globalise que la méthode de Shamanskii. Il serait possible d'adapter cette globalisation pour la direction de Chebyshev étant donné que cette direction utilise la même hessienne. L'idée de réutiliser la hessienne à plus d'une reprise pour la direction de Chebyshev est explorée dans [KD14]. Deuxièmement, la convergence de cet algorithme est intimement liée à la décomposition que l'on applique sur $\nabla^2 f(x)$ pour résoudre le système à la ligne 9. En effet, le critère pour déterminer si la décomposition n'est pas « trop différente » de $\nabla^2 f(x_k)$ est basé sur la décomposition LDL^T. Généralement, lorsque $\nabla^2 f(x)$ n'est pas définie positive, un multiple $\lambda > 0$ de la matrice identité est ajouté à D pour assurer que la matrice devienne définie positive. Si λ est considéré « trop grand », alors la décomposition LDL^T est «

trop différente » de la hessienne.

Les résultats numériques obtenus à l'aide de cette méthode se sont avérés mitigés. L'algorithme 6 a été codé tel qu'il est présenté dans [LS01], mais avec cette implémentation il est impossible de reproduire les résultats numériques mentionnés dans l'article. Il manque donc certains détails d'implémentation pour rendre cet algorithme compétitif avec les variantes que nous présentons dans les sections suivantes.

4.2 Globalisation par région de confiance

Les nouvelles idées présentées dans ce document portent sur la globalisation des méthodes d'ordre supérieur à travers un algorithme de région de confiance. Concrètement, nous voulons introduire des directions d'ordre supérieur dans l'algorithme 5. L'idée est de tirer profit des informations supplémentaires offertes par les directions d'ordre supérieur et ainsi avoir un algorithme qui converge

- en moins d'itérations que l'algorithme de région de confiance standard qui se réduit à la méthode de Newton ;
- de manière équivalente ou plus rapide en termes de temps de calcul.

Une autre comparaison que l'on pourrait faire est le nombre d'évaluations de fonction, gradient, etc. Cependant, nous considérons qu'un algorithme qui aurait besoin de moins d'évaluations de fonction, mais qui prendrait plus de temps pour atteindre une solution est moins intéressant qu'un algorithme qui prend plus d'évaluations de fonctions, mais moins de temps. C'est pour cette raison que le nombre d'itérations et le temps de calcul sont les mesures de performance utilisées.

Notons que les méthodes de Chebyshev, Halley et SuperHalley sont pénalisées par rapport à la méthode de Shamanskii à cause de la nécessité de calculer des dérivées d'ordre

trois. Quoique les résultats théoriques que nous présentons s'appliquent à toutes les méthodes d'ordre supérieur, les résultats numériques mettront l'accent sur la méthode de Shamanskii. Les résultats avec les méthodes de Chebyshev, Halley et SuperHalley seront à titre d'illustration. Pour des problèmes de « grande » dimension ($n > 40$) ces méthodes sont actuellement inutilisables à cause des limitations calculatoires imposées par les dérivées d'ordre trois.

Dans l'algorithme 5, l'objectif est de se ramener à la direction de Newton. Cette situation se produit lorsque $\lambda = 0$ lorsqu'on utilise les méthodes de décomposition (corollaire 1).

Nous étudions uniquement la globalisation par décomposition, c'est-à-dire l'approche de Moré-Sorensen pour résoudre (3.1).

Nous présentons différentes idées pour introduire les méthodes d'ordre supérieur dans l'algorithme 5. L'introduction de ces corrections d'ordre supérieur doit conserver certaines propriétés de l'algorithme 5 qui se réduit à la direction de Newton :

— $\{f(x_k)\}$ doit être monotone décroissante ;

— asymptotiquement, la convergence doit être cubique. Si on prend la direction de Newton, on se réduit à la convergence quadratique, donc avec les directions d'ordre supérieur la convergence locale devrait être au moins cubique.

Cette idée consiste à tenter de prendre la direction d'ordre supérieur lorsque trois conditions sont satisfaites :

1. $\lambda < \lambda_F$ où λ_F est une réel positif ou nul. Cette condition vise à prendre la direction d'ordre supérieur lorsque la direction produite naturellement par l'algorithme de région de confiance n'est pas trop différente de la direction de Newton ($\lambda = 0$).

2. $\|d_{\mathrm{HO}}\| \leqslant 2\Delta$; cette condition est importante pour assurer la convergence de l'algorithme. On remarque que la condition pourrait être $\|d_{\mathrm{HO}}\| \leqslant \kappa\Delta$ pour $\kappa \geqslant 1$ en vertu des conditions du théorème 3.3. Dans nos implémentations numériques

nous avons pris $\kappa = 2$.

3. la direction d'ordre supérieur doit produire une décroissance du modèle quadratique plus grande qu'une fraction de la décroissance du modèle quadratique produite par la direction produite naturellement par l'algorithme de région de confiance .

Les conditions 2 et 3 sont essentielles pour assurer que les conditions du théorème 3.3 sont satisfaites et ainsi assurer la convergence globale de l'algorithme 5 lorsque la direction d'ordre supérieur est choisie. Notons $r_\lambda = q(0) - q(d_\lambda)$ la décroissance du modèle quadratique lorsque la direction $d(\lambda)$ est choisie et posons $\eta_\lambda \in (0, 1)$. La direction de région de confiance d_{TR} choisie est telle que

$$d_{\text{TR}} = \begin{cases} d_{\text{HO}} \text{ si } \lambda \leqslant \lambda_F \textbf{ et } \|d_{\text{HO}}\| \leqslant 2\Delta \textbf{ et } q(0) - q(d_{\text{HO}}) \geqslant \eta_\lambda r_\lambda \\ d_N \text{ si } \lambda = 0 \text{ et que } d_{HO} \text{ n'est pas choisie} \\ d(\lambda^*) \text{ si } \lambda^* \neq 0 \text{ et que } d_{HO} \text{ n'est pas choisie.} \end{cases} \tag{4.2}$$

Une direction choisie avec (4.2) assure que l'on converge vers une solution. En effet, si

$$q(0) - q(d_{\text{HO}}) \geqslant \eta_\lambda r_\lambda \text{ et } r_\lambda \geqslant \eta_\lambda \kappa_{mdc} \tfrac{1}{2} \|\nabla f(x_k)\| \min\{\frac{\|\nabla f(x_k)\|}{\beta_k}, \Delta_k\}$$

alors

$$q(0) - q(d_{\text{HO}}) \geqslant \eta_\lambda \kappa_{mdc} \tfrac{1}{2} \|\nabla f(x_k)\| \min\{\frac{\|\nabla f(x_k)\|}{\beta_k}, \Delta_k\}.$$

On note que $\eta_\lambda \kappa_{mdc} \in (0, 1)$. Donc si d_{HO} est choisie, elle satisfait (3.4) et $\|d_{\text{HO}}\| \leqslant 2\Delta$, donc selon le théorème 3.3, l'algorithme 5 converge vers un point stationnaire lorsque (4.2) est utilisée pour choisir la direction. On remarque que les cas « d_N si $\lambda^* = 0$ » et « $d(\lambda^*)$ si $\lambda^* \neq 0$ » sont traités naturellement dans la version traditionnelle de l'algorithme 5.

Le choix des paramètres λ_F et η_λ influence la vitesse convergence de l'algorithme. Nous présentons différentes combinaisons de ces paramètres pour montrer comment ils influencent la convergence.

4.2.1 Décroissance du modèle quadratique ($\lambda_F = 0$ et $\eta_\lambda = 0$)

Avant d'en arriver à (4.2), nous avons effectué plusieurs tentatives avec d'autres stratégies pour introduire les directions d'ordre supérieur dans l'algorithme de région de confiance. Une des stratégies préliminaires testées consistait à toujours appliquer une correction d'ordre supérieur sur la direction de région de confiance produite naturellement par l'algorithme. Donc par exemple, si on prend la direction de Chebyshev et que l'on suppose la hessienne inversible on aurait

$$d_{TR} = d(\lambda) - \tfrac{1}{2}\nabla^2 f(x)^{-1}\nabla^3 f(x)d(\lambda)d(\lambda).$$

Cette idée produit de mauvais résultats numériques. En effet, loin d'une solution la correction d'ordre supérieur n'est pas nécessairement une correction. Dans la plupart des cas il s'agit bien souvent de « pollution » de la direction d_{TR}. Donc numériquement utiliser cette stratégie pour introduire de l'ordre supérieur dans l'algorithme de région de confiance ne converge presque jamais vers une solution. En plus d'être peu performante du point de vue numérique, cette manière de choisir la direction n'assure pas la convergence d'un point de vue théorique.

Une autre idée de globalisation est de prendre la correction d'ordre supérieur de d_{TR} seulement lorsqu'on est dans un voisinage de taille ε de la solution x^*, i.e.

$$d_{\mathrm{TR}} = \begin{cases} d_{\mathrm{HO}} \text{ si } \|\nabla f(x)\| < \varepsilon, \\ d(\lambda) \text{ sinon.} \end{cases} \tag{4.3}$$

D'un point de vue théorique, choisir la direction selon (4.3) converge puisque la correction

d'ordre supérieur est bien définie dans le voisinage d'une solution. Cependant, cette idée épargne peu ou pas d'itération. Il est possible que l'on sauve quelques itérations lorsqu'on est proche d'une solution, mais pas assez pour considérer qu'il s'agit d'une amélioration majeure de l'algorithme de région de confiance standard. Nous désirons obtenir une manière d'introduire l'ordre supérieur qui épargne de nombreuses itérations (et idéalement du temps de calcul). Un autre inconvénient de cette idée est que ε est différent pour chaque f.

La première idée fructueuse pour assurer des progrès significatifs à l'algorithme de région de confiance consiste a prendre $\lambda_F = 0$ et $\eta_\lambda = 0$ dans (4.2). Concrètement, cela correspond à considérer la direction d'ordre supérieur lorsque la direction de Newton est la direction naturellement produite par l'algorithme de région de confiance et que la direction d'ordre supérieur produit une décroissance du modèle quadratique q_x :

$$d_{\text{TR}} = \begin{cases} d_{\text{HO}} \text{ si } \lambda^* = 0 \textbf{ et } \|d_{\text{HO}}\| \leqslant 2\Delta \textbf{ et } q(0) - q(d_{\text{HO}}) \geqslant 0 \\ d_N \text{ si } \lambda^* = 0 \text{ et } d_{\text{HO}} \text{ n'est pas choisie,} \\ d(\lambda^*) \text{ si } \lambda^* \neq 0 \text{ et } d_{\text{HO}} \text{ n'est pas choisie..} \end{cases}$$

Le choix de λ_F influence la performance de l'algorithme, particulièrement en termes de temps de calcul. En effet, d'un point de vue d'implémentation numérique, la direction d'ordre supérieur n'est pas toujours calculée. Elle n'est calculée que si $\lambda \leqslant \lambda_F$. Les autres critères de sélection sont considérés une fois que d_{HO} est calculée.

Si $\lambda_F = 0$, alors la direction d'ordre supérieur n'est considérée que si $\lambda^* = 0$, donc elle n'est pas calculée à de nombreuses reprises. En effet, en prenant plutôt $\lambda_F = \infty$, la direction d'ordre supérieur serait calculée à chaque itération, ralentissant considérablement l'algorithme. Le choix de $\lambda_F = 0$ permet donc de minimiser les coûts de calcul supplémentaires de la direction d'ordre supérieur.

Le choix de η_λ n'influence pas la performance de l'algorithme. En effet, une fois que la direction d'ordre supérieur est calculée, les quantités $q(0) - q(d_{\text{HO}})$ et r_λ ne représentent

pas un calcul dont le coût est significatif.

Exemple numérique

Nous regardons un exemple numérique pour comparer un algorithme de région de confiance classique qui utilise l'approche de Moré-Sorensen (`TR_Nwt`) à la variante qui utilise (4.2) avec λ_F et η_λ nuls pour choisir une direction d'ordre supérieur. En plus des corrections d'ordre supérieur classiques, nous regardons également la correction Shamanskii-BFGS proposée à la section 2.5. La fonction de Rosenbrock (1.10) est utilisée pour comparer les directions d'ordre supérieur avec $x_0 = (-1.2, 1.0)$. Les résultats sont consignés dans le tableau 4.1.

Nous remarquons que l'introduction des méthodes d'ordre supérieur dans l'algorithme 5 avec (4.2) où λ_F et η_λ sont nuls permet d'économiser des itérations par rapport à l'algorithme de région de confiance traditionnel. À l'exception de Shamanskii-BFGS, prendre une direction d'ordre supérieur économise au moins deux itérations. Dans cet exemple, la direction de Chebyshev est la plus efficace, elle prend 12 itérations de moins qu'un algorithme de région de confiance traditionnel. Finalement, nous observons qu'asymptotiquement les directions d'ordre supérieur exhibent des propriétés de convergence cubique.

Donc (4.2) avec $\eta_\lambda = \lambda_F = 0$ peut améliorer l'algorithme de région de confiance classique. Cependant, le choix de ces paramètres n'est pas toujours idéal. En effet, le tableau 4.2 présente l'algorithme de région de confiance sur la fonction de Rosenbrock avec le point de départ $x_0 = (\frac{1}{3}, \frac{2}{3})$ ainsi que la version de qui choisi la direction de Shamanskii lorsque $\eta_\lambda = \lambda_F = 0$. On remarque que 11 itérations sont nécessaires pour choisir la direction d'ordre supérieur dans l'algorithme traditionnel, contre 14 pour la version qui utilise parfois la direction de Shamanskii. On remarque que la direction d'ordre supérieur (Shamanskii) est utilisée aux itérations 5, 6 et à partir de l'itération 10 jusqu'à la quatorzième. Donc, un point stationnaire a été atteint, mais un algorithme de région de

Itérations	Région de confiance Newton	Shamanskii	Sham.-BFGS	Chebyshev	Halley	SHalley
0	2.3×10^{2}	2.3×10^{2}	2.3×10^{2}	2.3×10^{2}	2.3×10^{2}	2.3×10^{2}
1	4.6×10^{0}	5.1×10^{0}	4.6×10^{0}	5.1×10^{0}	5.9×10^{0}	4.6×10^{0}
2	4.6×10^{0}	5.1×10^{0}	1.7×10^{0}	5.1×10^{0}	4.9×10^{0}	4.6×10^{0}
3	4.6×10^{0}	5.1×10^{0}	1.7×10^{0}	5.1×10^{0}	4.9×10^{0}	4.6×10^{0}
...	...	...	...	...	...	...
19	9.0×10^{-1}	7.3×10^{0}	7.6×10^{0}	3.7×10^{0}	4.6×10^{0}	9.0×10^{-01}
20	9.0×10^{-1}	1.5×10^{0}	4.1×10^{0}	6.0×10^{-1}	5.1×10^{0}	9.0×10^{-01}
21	9.0×10^{-1}	6.6×10^{0}	2.1×10^{0}	2.8×10^{-1}	7.1×10^{0}	9.0×10^{-01}
22	9.0×10^{-1}	8.7×10^{-1}	2.2×10^{-1}	8.0×10^{-4}	2.8×10^{0}	9.0×10^{-01}
23	1.2×10^{0}	2.6×10^{0}	1.6×10^{0}	9.4×10^{-10}	9.8×10^{0}	1.2×10^{00}
24	1.2×10^{0}	2.6×10^{0}	8.1×10^{-1}	3.7×10^{-29}	3.4×10^{0}	1.2×10^{00}
25	8.2×10^{-1}	2.6×10^{0}	1.4×10^{0}	2.0×10^{-70}	2.2×10^{0}	8.2×10^{-01}
26	2.8×10^{0}	2.6×10^{0}	9.6×10^{-1}		3.2×10^{0}	2.8×10^{0}
27	2.0×10^{0}	1.9×10^{-1}	1.2×10^{0}		1.3×10^{0}	2.0×10^{0}
28	2.3×10^{00}	1.7×10^{0}	1.0×10^{0}		2.5×10^{0}	2.3×10^{0}
29	9.6×10^{-01}	9.1×10^{-2}	1.1×10^{0}		9.1×10^{-1}	9.6×10^{-1}
30	9.7×10^{-01}	1.4×10^{-3}	1.0×10^{0}		2.8×10^{-2}	9.7×10^{-1}
31	1.5×10^{-01}	3.5×10^{-9}	1.1×10^{0}		4.6×10^{-6}	1.1×10^{-1}
32	4.0×10^{-02}	2.9×10^{-26}	1.0×10^{0}		6.1×10^{-19}	1.4×10^{-5}
33	2.6×10^{-04}	0.0×10^{0}	9.8×10^{-1}		9.5×10^{-55}	1.9×10^{-12}
34	1.2×10^{-07}		9.1×10^{-1}			3.4×10^{-34}
35	2.5×10^{-15}		7.9×10^{-1}			0.0×10^{00}
36	1.1×10^{-29}		5.2×10^{-1}			
37	2.0×10^{-59}		1.1×10^{-1}			
38			5.5×10^{-4}			
39			5.3×10^{-11}			
40			5.3×10^{-32}			
41			3.5×10^{-75}			

Tableau 4.1 – Comparaison de $\|\nabla f(x)\|$ avec des méthodes d'ordre supérieur globalisées appliquées à la fonction de Rosenbrock (point de départ $(-1.2, 1.0)$).

confiance traditionnel l'atteint en 3 itérations de moins.

Région de confiance				Région de confiance avec Shamanskii		
k	$\|\nabla f(x)\|$	λ	Direction Choisie	$\|\nabla f(x)\|$	λ	Direction Choisie
0	1.3×10^{2}	0.0×10^{0}		1.3×10^{2}	0.0×10^{0}	d_{TR}
1	1.3×10^{2}	1.4×10^{2}	d_{TR}	1.3×10^{2}	1.4×10^{2}	d_{TR}
2	1.3×10^{2}	2.1×10^{2}	d_{TR}	1.3×10^{2}	2.1×10^{2}	d_{TR}
3	7.1×10^{1}	1.8×10^{3}	d_{TR}	7.1×10^{1}	1.8×10^{3}	d_{TR}
4	2.1×10^{1}	1.4×10^{2}	d_{TR}	2.1×10^{1}	1.4×10^{2}	d_{TR}
5	1.3×10^{-01}	0.0×10^{0}	d_{TR}	9.1×10^{-2}	0.0×10^{0}	d_{HO}
6	1.9×10^{0}	0.0×10^{0}	d_{TR}	1.5×10^{0}	0.0×10^{0}	d_{HO}
7	4.8×10^{-04}	0.0×10^{0}	d_{TR}	1.5×10^{0}	0.0×10^{0}	d_{TR}
8	2.6×10^{-05}	0.0×10^{0}	d_{TR}	1.5×10^{0}	0.0×10^{0}	d_{TR}
9	8.4×10^{-14}	0.0×10^{0}	d_{TR}	2.0×10^{0}	2.7×10^{-1}	d_{TR}
10	7.9×10^{-25}	0.0×10^{0}	d_{TR}	5.2×10^{-2}	0.0×10^{0}	d_{HO}
11	8.0×10^{-53}	0.0×10^{0}	d_{TR}	3.1×10^{-4}	0.0×10^{0}	d_{HO}
12				4.1×10^{-11}	0.0×10^{0}	d_{HO}
13				4.4×10^{-32}	0.0×10^{0}	d_{HO}
14				6.9×10^{-75}	0.0×10^{0}	d_{HO}

Tableau 4.2 – Algorithmes de région de confiance appliqué à la fonction de Rosenbrock (point de départ $(\frac{1}{3}, \frac{2}{3})$)

4.2.2 Comparaison de décroissance du modèle quadratique ($\eta_\lambda > 0$)

Pour minimiser les risques que la stratégie (4.2) pour choisir la direction de descente pénalise l'algorithme de région de confiance, il est possible de prendre un $\eta_\lambda > 0$ dans (4.2).

Comme nous l'avons mentionné à la section précédente, le coût de calcul dépend de

λ_F, donc tant que λ_F est nul, prendre $\eta_\lambda > 0$ n'augmente pas le temps de calcul de l'algorithme 5. Le choix de η_λ reste délicat. Si η_λ est trop proche de zéro, cela peut conduire à des situations où prendre la direction d'ordre supérieur ralentit l'algorithme de région de confiance comme nous venons de le voir. Cependant, si η_λ est trop proche de 1 alors, la direction d'ordre supérieur n'est presque jamais choisie et donc il n'y a aucun bénéfice à utiliser (4.2).

Exemple numérique

Nous présentons un exemple numérique pour illustrer la convergence de l'algorithme 5 lorsque (4.2) est utilisée avec $\lambda_F = 0$ et $\eta_\lambda = 0.25$ pour choisir la direction d'ordre supérieur. La fonction de Rosenbrock est utilisée avec le point de départ $x_0 = (\frac{1}{3}, \frac{2}{3})$. Les résultats sont consignés dans le tableau 4.3.

Itérations	$\|\nabla f(x)\|$	λ	Direction choisie
0	1.3×10^{2}	0.0×10^{0}	
1	1.3×10^{2}	1.4×10^{2}	d_{TR}
2	1.3×10^{2}	2.1×10^{2}	d_{TR}
3	7.1×10^{1}	1.8×10^{3}	d_{TR}
4	2.1×10^{1}	1.4×10^{2}	d_{TR}
5	9.1×10^{2}	0.0×10^{0}	d_{HO}
6	1.6×10^{0}	0.0×10^{0}	d_{TR}
7	2.8×10^{-4}	0.0×10^{0}	d_{HO}
8	2.3×10^{-9}	0.0×10^{0}	d_{HO}
9	8.2×10^{-24}	0.0×10^{0}	d_{HO}
10	1.3×10^{-67}	0.0×10^{0}	d_{HO}

Tableau 4.3 – Algorithme de région de confiance appliqué qui utilise la direction de Shamanskii et comparaison avec la décroissance quadratique du modèle de Newton appliqué à la fonction de Rosenbrock.

On voit donc que la stratégie (4.2) avec $\lambda_F = 0$ et $\eta_\lambda = 0.25$ pour choisir la direction d'ordre supérieur est une amélioration de cette même stratégie lorsque les deux paramètres sont nuls. En effet, seulement 10 itérations sont nécessaires contrairement à 14, cela représente également une itération de moins que l'algorithme `TR_Nwt`. On remarque dans le tableau 4.3 que la direction d'ordre supérieur est choisie à l'itération 5, mais pas à l'itération 6. Tandis que dans le tableau 4.3 la direction d'ordre supérieur était choisie pour les itérations 5 et 6. Donc, pour cet exemple, utiliser (4.2) améliore l'algorithme de région de confiance.

4.2.3 Direction d'ordre supérieur avec $\lambda \neq 0$ et $\eta_\lambda \neq 0$

Tel que mentionné dans les sections précédentes, le choix de λ_F est important, car un λ_F grand signifie que la direction d_{HO} est calculée plus souvent. Cependant, il peut y avoir des bénéfices à utiliser un $\lambda_F \neq 0$. En effet, il est possible que la correction d'ordre supérieur améliore la direction de région de confiance, même s'il ne s'agit pas de la direction de Newton.

Prenons par exemple la fonction de Bard [MGH81] définie par (4.4) avec comme point de départ $x_0 = (1, 1, 1)$.

$$f(x) = \sum_{j=1}^{15} f_j(x) \tag{4.4}$$

$$\begin{aligned}
\text{où } f_j(x) &= y_j - \Big(x_1 + \big(\tfrac{u_j}{v_j x_2 + w_j x_3}\big)\Big), \\
\text{avec } y &= (0.14, 0.18, 0.22, 0.25, 0.29, 0.32, 0.35, 0.39, 0.37, 0.58, 0.73, 0.96, 1.34, 2.10, 4.39), \\
u &= (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15), \\
v &= (15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1),
\end{aligned} \tag{4.5}$$

Dans le tableau 4.4, on peut voir qu'une région de confiance traditionnelle nécessite 32 itérations pour atteindre un point stationnaire. On peut également observer les résultats de l'algorithme 5 avec la stratégie (4.2) qui utilise $\lambda_F = 0$ et $\eta_\lambda = 0.8$. On remarque qu'aucune itération n'est épargnée. Le comportement de l'algorithme est exactement le même qu'avec un algorithme de région de confiance.

Région de confiance				Région de confiance avec Shamanskii		
k	$\|\nabla f(x)\|$	λ	Direction Choisie	$\|\nabla f(x)\|$	λ	Direction Choisie
0	8.5×10^{1}	0.0×10^{0}		8.5×10^{1}	0.0×10^{0}	
1	1.2×10^{1}	4.1×10^{-03}	d_{TR}	1.2×10^{1}	4.1×10^{-03}	d_{TR}
2	1.2×10^{1}	2.1×10^{-02}	d_{TR}	1.2×10^{1}	2.1×10^{-02}	d_{TR}
3	1.2×10^{1}	4.1×10^{-02}	d_{TR}	1.2×10^{1}	4.1×10^{-02}	d_{TR}
4	9.9×10^{0}	3.8×10^{0}	d_{TR}	9.9×10^{0}	3.8×10^{0}	d_{TR}
5	9.9×10^{0}	8.6×10^{-02}	d_{TR}	9.9×10^{0}	8.6×10^{-02}	d_{TR}
6	8.9×10^{0}	7.8×10^{0}	d_{TR}	8.9×10^{0}	7.8×10^{0}	d_{TR}
7	8.9×10^{0}	4.2×10^{0}	d_{TR}	8.9×10^{0}	4.2×10^{0}	d_{TR}
8	8.9×10^{0}	2.2×10^{1}	d_{TR}	8.9×10^{0}	2.2×10^{1}	d_{TR}
9	8.8×10^{0}	2.1×10^{2}	d_{TR}	8.8×10^{0}	2.1×10^{2}	d_{TR}
10	8.5×10^{0}	2.7×10^{1}	d_{TR}	8.5×10^{0}	2.7×10^{1}	d_{TR}
...	...	...	...			
30	1.9×10^{-04}	0.0×10^{0}	d_{TR}	1.9×10^{-04}	0.0×10^{0}	d_{TR}
31	6.5×10^{-06}	0.0×10^{0}	d_{TR}	6.5×10^{-06}	0.0×10^{0}	d_{TR}
32	2.9×10^{-10}	0.0×10^{0}	d_{TR}	2.9×10^{-10}	0.0×10^{0}	d_{TR}

Tableau 4.4 – Algorithme de région de confiance appliqué à la fonction de Bard.

Or, si on prend $\lambda_F = 1000$ et $\eta_\lambda = 0.8$, plus de 20 itérations sont épargnées comme il est possible de voir dans le tableau 4.5. On remarque également que la direction d'ordre supérieur est toujours choisie, contrairement au tableau 4.4 où la direction d'ordre supérieur n'est jamais choisie. Donc, il peut être avantageux de considérer la direction

d'ordre supérieur même lorsque la direction de Newton n'est pas la direction produite naturellement par l'algorithme de région de confiance.

Itérations	$\|\nabla f(x)\|$	λ	Direction choisie
0	8.5×10^{01}	0.0×10^{00}	
1	8.6×10^{00}	4.1×10^{-03}	d_{HO}
2	2.2×10^{00}	0.0×10^{00}	d_{HO}
3	3.8×10^{-01}	0.0×10^{00}	d_{HO}
4	6.6×10^{-02}	0.0×10^{00}	d_{HO}
5	4.5×10^{-02}	0.0×10^{00}	d_{HO}
6	2.0×10^{-02}	0.0×10^{00}	d_{HO}
7	2.4×10^{-03}	0.0×10^{00}	d_{HO}
8	3.3×10^{-06}	0.0×10^{00}	d_{HO}
9	6.0×10^{-15}	0.0×10^{00}	d_{HO}

Tableau 4.5 – Algorithme de région de confiance globalisé avec $(\lambda_F, \eta_\lambda) = (1000.0, 0.8)$ appliqué à la fonction de Bard.

Il n'existe pas de combinaisons λ_F et η_λ qui améliorent systématiquement l'algorithme de région de confiance. Cependant nous verrons à la section 4.4 qu'en général la combinaison $(\lambda_F, \eta_\lambda) = (1000, 0.8)$ est efficace.

4.3 Globalisation par les méthodes ARC

Comme nous l'avons vu à la section 3.3, lorsque l'approche de Moré-Sorensen est utilisée pour résoudre (3.9) le paramètre λ ne se réduit jamais à zéro. Or, à la section précédente nous avons vu que la stratégie (4.2) assure la convergence de l'algorithme 5 même si on prend la direction d'ordre supérieur lorsque $\lambda \neq 0$. Il est donc naturel de vouloir regarder si cette stratégie améliore également la stratégie ARC traditionnelle. Posons $r_{\text{ARC}} = q(0) - q(d_{\text{ARC}})$ la décroissance du modèle quadratique de la direction produite

naturellement par l'algorithme ARC. Nous notons h_{HO} la correction d'ordre supérieure appliquée à une direction d. Par exemple pour la direction de Shamanskii, $h_{\text{HO}} = d_{\tilde{S}}$ calculée à la section 2.3. Nous proposons donc la stratégie suivante pour introduire des directions d'ordre supérieur dans l'algorithme ARC :

$$d_{\text{ARC}} = \begin{cases} d_{\text{HO}} \text{ si } \lambda^* \leqslant \lambda_F \textbf{ et } \|h_{\text{HO}}\| \leqslant 2\|d(\lambda^*)\| \textbf{ et } q(0) - q(d_{\text{HO}}) \geqslant \eta_\lambda r_{\text{ARC}}, \\ d(\lambda^*) \text{ si } \lambda^* \neq 0. \end{cases} \tag{4.6}$$

Cette stratégie rappelle fortement (4.2). Nous rappelons qu'avec la stratégie ARC, λ ne se réduit jamais à zéro. Nous remarquons que le critère $\|h_{\text{HO}}\| \leqslant 2\|d(\lambda^*)\|$ est différent de celui présenté dans (4.2). Ce critère est très important pour démontrer que le choix de d_{ARC} par (4.6) ne modifie pas les propriétés de convergence globale de l'algorithme ARC.

Avant de présenter un exemple numérique, nous justifions que choisir la direction avec la stratégie (4.6) garantit la convergence de l'algorithme ARC. Comme nous l'avons vu à la section 3.3, nous utilisons la variante ARCq de l'algorithme ARC proposée dans [Dus18]. Nous proposons d'adapter le théorème 3.6 et sa preuve pour $\{x_k\}$ généré par l'algorithme 5 dans sa version ARC qui utilise (4.6) pour choisir la direction.

Théorème 4.1 *Soit $\{x_k\}$ la suite de points engendrés par l'algorithme 5 dans sa version ARC qui utilise (4.6) pour choisir la direction. Soit $f \in \mathcal{C}^2(\mathbb{R}^n)$ telle que $f(x_k)$ est bornée inférieurement et que $\{x_k\}$ demeure dans un compact $C \subset \mathbb{R}^n$, alors il existe une sous-suite $\{x_{k_i}\}$, extraite de $\{x_k\}$, qui converge vers un point d'accumulation qui satisfait aux conditions nécessaires d'ordre deux.*

Plus précisément, pour toute sous-suite $\{x_{k_i}\}$, extraite de $\{x_k\}$, qui converge vers un point d'accumulation x^*, ce point d'accumulation satisfait aux conditions nécessaires d'ordre deux.

Démonstration. Tout d'abord, nous notons la direction utilisée $d_k = d_{\text{ARC}_k} + h_k$ où

d_{ARC} est la direction naturellement produite par l'algorithme ARC et h_k est la correction d'ordre supérieur appliquée à d_{ARC_k}. Si la direction d'ordre supérieur n'a pas été choisie à l'itération k alors $h_k = 0$.

Les premiers développements de la preuve sont identiques à ceux de la preuve du théorème 3.6. Donc $\{x_{k_i}\}$ est une sous-suite de $\{x_k\}$ qui converge vers un point d'accumulation $\bar{x}$. De plus, avec des arguments identiques à la preuve du théorème 3.6, nous pouvons monter que $\Delta f(x_k) \to 0$ et $\Delta q(d_k) \to 0$.

On pose également les mêmes quantités que pour la preuve du théorème 3.6 :

— $\bar{d}$ une solution optimale de $\min_{\bar{x}}^{\bar{\alpha}}$;

— $\hat{x} = \bar{x} + \bar{d}$;

— $\hat{d_{k_i}} = \hat{x} - x_{k_i}$.

Pour utiliser un raisonnement similaire à celui présenté dans la preuve du théorème 3.6, nous devons montrer que certaines quantités tendent vers zéro.

Si $d_k = d_{\text{HO}_k}$, cela signifie que $\Delta q(d_{k_i}) = \Delta q(d_{HO_{k_i}}) \geq \eta \Delta q(d_{\text{ARC}_{k_i}})$. Donc si $\Delta q(d_{k_i}) \to 0$ alors $\Delta q(d_{\text{ARC}_{k_i}}) \to 0$ par le théorème des gendarmes.

De plus, sachant que $\Delta q(d_{\text{ARC}_{k_i}}) \to 0$ on peut montrer que $\Delta c_{x_k}^{\alpha_k}(d_{\text{ARC}_{k_i}}) \to 0$. En effet,

$$0 \geq -\Delta c_{x_k}^{\alpha_k}(d_{\text{ARC}_k}) \geq -\Delta q_{x_k}(d_{\text{ARC}_k}) \tag{4.7}$$

donc à la limite $-\Delta c_{x_k}^{\alpha_k}(d_{\text{ARC}_k}) \to 0$ d'où $\Delta c_{x_k}^{\alpha_k}(d_{\text{ARC}_k}) \to 0$.

De plus, il est important de montrer que $d_{\text{ARC}_{k_i}} \to 0$. En effet, si on reprend (4.7) en substituant $-\Delta c_{x_k}^{\alpha_k}(d_{\text{ARC}_k})$ par $-\Delta q_{x_k}(d_{\text{ARC}_k}) + \frac{\|d_{\text{ARC}_k}\|^3}{3\alpha_k}$ on obtient :

$$0 \geq -\Delta q_{x_k}(d_{\text{ARC}_k}) + \frac{\|d_{\text{ARC}_k}\|^3}{3\alpha_k} \geq -\Delta q_{x_k}(d_{\text{ARC}_k}).$$

Considérant que $\Delta q(d_{\text{ARC}_{k_i}}) \to 0$, on a que $\frac{\|d_{\text{ARC}_k}\|^3}{3\alpha_k} \to 0$ et comme $\alpha_k \geqslant \bar{\alpha} > 0$ la direction $d_{\text{ARC}_{k_i}}$ tend vers zéro.

La direction d'ordre supérieur est choisie lorsque $\|h_{\mathrm{HO}_k}\| \leqslant 2\|d_{\mathrm{ARC}_k}\|$, donc si $d_{\mathrm{ARC}_{k_i}} \to 0$ alors $h_{\mathrm{HO}_{k_i}} \to 0$.

Maintenant, nous construisons une équation similaire à (3.14). On a que :

$$\begin{aligned}
\Delta c_{x_k}^{\alpha_k}(d_k) = &\ \Delta c_{x_k}^{\alpha_k}(d_{\mathrm{ARC}_k} + h_k) \\
= &\ c_{x_k}^{\alpha_k}(0) - c_{x_k}^{\alpha_k}(d_{\mathrm{ARC}_k} + h_k) \\
= &\ f(x_k) - (f(x_k) + \nabla f(x_k)(d_{\mathrm{ARC}_k} + h_k) \\
&\ + (d_{\mathrm{ARC}_k} + h_k)^t \nabla^2 f(x_k)(d_{\mathrm{ARC}_k} + h_k) + \frac{\|d_{\mathrm{ARC}_k} + h_k\|^3}{3\alpha_k}) \\
= &\ f(x_k) - (f(x_k) + \nabla f(x_k) d_{\mathrm{ARC}_k} + d_{ARC_k}^t \nabla^2 f(x_k) d_{\mathrm{ARC}_k} + \frac{\|d_{\mathrm{ARC}_k}\|}{3\alpha_k} \\
&\ + \nabla f(x_k) h_k + h_k^t \nabla^2 f(x_k) h_k + \frac{\|h_k\|^3}{3\alpha_k} + \frac{\|d_{\mathrm{ARC}_k}\|^2 \|h_k\| + \|d_{\mathrm{ARC}_k}\| \|h_k\|^2}{\alpha_k}) \\
&\ + 2 d_{ARC_k}^t \nabla^2 f(x_k) h_k \\
= &\ f(x_k) - c_{x_k}^{\alpha_k}(d_{\mathrm{ARC}_k}) + \Delta c_{x_k}^{\alpha_k}(h_k) \\
&\ - (\frac{\|d_{\mathrm{ARC}_k}\|^2 \|h_k\| + \|d_{\mathrm{ARC}_k}\| \|h_k\|^2}{\alpha_k} + 2 d_{\mathrm{ARC}_k}^t \nabla^2 f(x_k) h_k).
\end{aligned}$$

Donc en posant

$$\delta_k = \Delta c_{x_k}^{\alpha_k}(h_k) - (\frac{\|d_{\mathrm{ARC}_k}\|^2 \|h_k\| + \|d_{\mathrm{ARC}_k}\| \|h_k\|^2}{\alpha_k} + 2 d_{\mathrm{ARC}_k}^t \nabla^2 f(x_k) h_k),$$

on obtient

$$c_{x_k}^{\alpha_k}(d_{\mathrm{ARC}_k}) = f(x_k) - \Delta c_{x_k}^{\alpha_k}(d_k) + \delta_k.$$

Considérant d_{ARC_k} le minimum de $c_{x_k}^{\alpha_k}$, l'équivalent de (3.14) est :

$$m_{x_{k_i}}^{\bar{\alpha}}\left(\hat{d}_{k_i}\right) \geq m_{x_{k_i}}^{\alpha_{k_i}}\left(\hat{d}_{k_i}\right) \geq m_{x_{k_i}}^{\alpha_{k_i}}\left(d_{ARC_{k_i}}\right) = f\left(x_{k_i}\right) - \Delta m_{x_{k_i}}^{\alpha_{k_i}}(d_{HO_{k_i}}) + \delta_{k_i}. \tag{4.8}$$

En prenant les limites des termes de gauche et de droite de (4.8) on obtient :

$$c_{\bar{x}}^{\bar{\alpha}}(\bar{d}) \geq f(\bar{x}) = c_{\bar{x}}^{\bar{\alpha}}(0).$$

Puisque 0 est un minimum global de $c_{\bar{x}}^{\bar{\alpha}}$ selon le théorème 3.5 $\bar{x}$ satisfait aux conditions nécessaires d'ordre deux. ■

Exemple numérique

Nous présentons un exemple numérique pour justifier que (4.6) peut bel et bien améliorer la performance de la stratégie ARC. La fonction de Rosenbrock avec le point de départ $x_0 = (\frac{1}{3}, \frac{2}{3})$ est utilisée. Les résultats de l'algorithme ARC sont consignés dans le tableau 4.6 ainsi que ceux utilisant (4.6) avec $\lambda_F = 100.0$ et $\eta_\lambda = 0.25$.

On voit que l'algorithme ARC nécessite 18 itérations pour atteindre un point stationnaire. Lorsque (4.6) est utilisée pour déterminer si la direction d'ordre supérieur est choisie, l'algorithme nécessite 15 itérations, soit 3 de moins que la version standard. On remarque qu'aux itérations 7 et 8 on a $\lambda \leqslant 100$, mais que la direction d'ordre supérieur n'est pas choisie. Cependant, la direction d'ordre supérieur est choisie pour toutes les itérations à partir de l'itération 9. On peut observer la convergence cubique lorsque la direction d'ordre supérieur est choisie à l'aide de la norme du gradient.

Il est important de mentionner que le gain d'itérations est souvent asymptotique, et ce, pour les méthodes de région de confiance ou de régularisation cubique adaptative. Cependant, la direction d'ordre supérieur n'est pas toujours choisie en régime asymptotique, par exemple la direction de Chebyshev permet un gain d'itération loin de la solution dans le tableau 4.1. Une analyse plus raffinée des paramètres λ_F et η_λ pourrait permettre de déterminer comment optimiser l'utilisation des corrections d'ordre supérieur loin d'un minimum local.

ARC				ARC avec Shamanskii		
k	$\|\nabla f(x)\|$	λ	Direction Choisie	$\|\nabla f(x)\|$	λ	Direction Choisie
0	1.3×10^{2}			1.3×10^{2}	0.0×10^{0}	d_{ARC}
1	1.3×10^{2}	1.3×10^{2}	d_{ARC}	1.3×10^{2}	1.3×10^{2}	d_{ARC}
2	1.3×10^{2}	1.3×10^{2}	d_{ARC}	1.3×10^{2}	1.3×10^{2}	d_{ARC}
3	1.3×10^{2}	1.3×10^{2}	d_{ARC}	1.3×10^{2}	1.3×10^{2}	d_{ARC}
4	1.3×10^{2}	1.5×10^{2}	d_{ARC}	1.3×10^{2}	1.5×10^{2}	d_{ARC}
5	1.2×10^{2}	4.0×10^{2}	d_{ARC}	1.2×10^{2}	4.0×10^{2}	d_{ARC}
6	3.2×10^{1}	1.4×10^{2}	d_{ARC}	3.2×10^{1}	1.4×10^{2}	d_{ARC}
7	2.0×10^{0}	1.6×10^{1}	d_{ARC}	2.0×10^{0}	1.6×10^{1}	d_{ARC}
8	5.5×10^{-1}	2.6×10^{0}	d_{ARC}	5.5×10^{-1}	2.6×10^{0}	d_{ARC}
9	2.0×10^{0}	9.2×10^{-1}	d_{ARC}	1.0×10^{0}	9.2×10^{-1}	d_{HO}
9	2.0×10^{0}	9.2×10^{-1}	d_{ARC}	1.0×10^{0}	9.2×10^{-1}	d_{HO}
10	8.6×10^{-1}	1.5×10^{-1}	d_{ARC}	3.9×10^{-1}	1.6×10^{-1}	d_{HO}
11	9.4×10^{-1}	3.0×10^{-2}	d_{ARC}	1.7×10^{-2}	2.1×10^{-2}	d_{HO}
12	1.4×10^{-1}	2.7×10^{-3}	d_{ARC}	1.2×10^{-4}	1.1×10^{-3}	d_{HO}
13	3.6×10^{-2}	2.5×10^{-4}	d_{ARC}	9.5×10^{-10}	3.6×10^{-6}	d_{HO}
14	2.2×10^{-4}	4.3×10^{-6}	d_{ARC}	1.7×10^{-25}	4.6×10^{-40}	d_{HO}
15	8.8×10^{-8}	1.6×10^{-8}	d_{ARC}	1.1×10^{-73}	4.6×10^{-40}	d_{HO}
16	1.3×10^{-15}	4.6×10^{-40}	d_{ARC}			
17	3.0×10^{-30}	4.6×10^{-40}	d_{ARC}			
18	1.5×10^{-60}	4.6×10^{-40}	d_{ARC}			

Tableau 4.6 – Algorithme ARC appliqué à la fonction de Rosenbrock.

4.4 Comparaison d'algorithmes et discussion

Après avoir regardé diverses stratégies sur des exemples précis, il est intéressant de regarder leur performance sur une vaste collection de problèmes. Cependant, il y a certaines considérations numériques à prendre en compte.

Tout d'abord, les `BigFloat` ne peuvent pas être utilisés par de nombreux outils numériques. En effet, de nombreux outils sont optimisés en simple précision (`Float32`) ou en double précision (`Float64`) et ne fonctionnent pas avec d'autres précisions. C'est le cas des problèmes de la collection `CUTEst` [GOT15], l'outil de modélisation mathématique JuMP [LD15] ou encore les routines d'algèbre linéaire HSL [HSL07].

Les directions d'ordre supérieur bénéficient souvent de l'usage des `BigFloat`, car la convergence asymptotique de ces directions est cubique, il est plus rapide d'aller chercher une grande précision grâce à ces directions. Or les outils à notre disposition qui travaillent en `BigFloat` n'utilisent que des matrices denses. Avec des matrices denses, les allocations de mémoire deviennent rapidement un problème. Cela limite donc le nombre de variables de notre problème si on désire utiliser les `BigFloat`.

De plus, comme nous l'avons indiqué aux sections 2.2.1 et 2.2.2, le calcul des dérivées d'ordre trois est limité lui aussi par n et les outils de DA à notre disposition. Donc, pour illustrer le comportement des méthodes de Chebyshev, Halley et SuperHalley on se limite à $n < 5$ peu importe la précision utilisée.

4.4.1 Comparaison d'algorithmes en BigFloat

Pour comparer différentes stratégies de globalisation des méthodes d'ordre supérieur nous utilisons une collection de 63 problèmes à dimensions variables. Puisque nous travaillons avec des `BigFloat` nous bornons supérieurement n par 5. La précision recherchée est

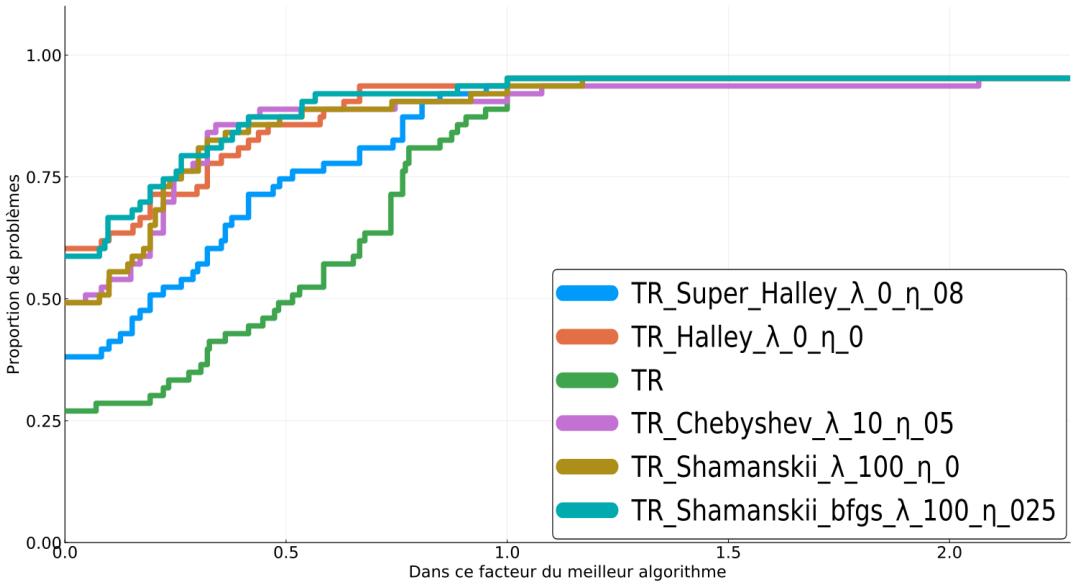

Figure 4.1 – Comparaison de l'algorithme de région de confiance et différentes stratégies de globalisation de méthodes d'ordre supérieur en fonction du nombre d'itérations en BigFloat

de 10^{-40} lorsque les nombres sont représentées en `BigFloat`, autrement dit on considère qu'un point stationnaire est atteint lorsque $\|\nabla f(x)\| \leqslant 10^{-40}$.

Des comparaisons en termes de nombre d'itérations et de temps de calcul sur une sélection de 63 problèmes des collections Moré-Garbow-Hillstrom [MGH81] et `CUTEst` [GOT15] sont présentées dans les figures 4.1 et 4.2 respectivement.

En termes d'itérations on remarque que toutes les globalisations de méthodes d'ordre supérieur sont plus rapides ou équivalentes à la méthode de région de confiance classique, et ce, peu importe le choix des paramètres λ_F et η_λ. De plus, toutes les méthodes de globalisation résolvent le même nombre de problèmes.

Cependant, le temps de calcul des méthodes de Chebyshev et (Super)Halley est nettement supérieur à la méthode classique et aux globalisations de la direction de Shamanskii.

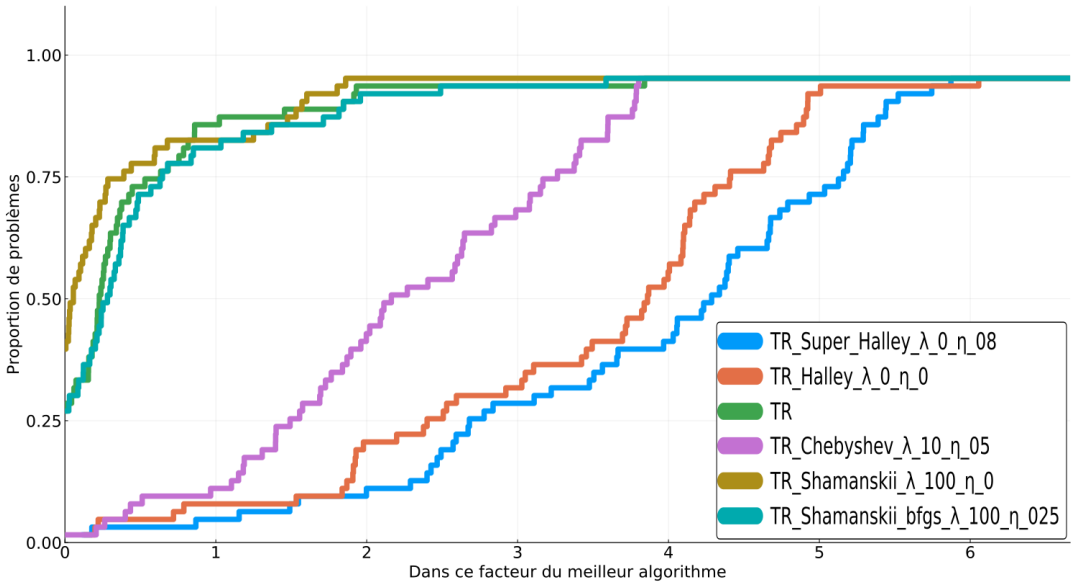

Figure 4.2 – Comparaison de l'algorithme de région de confiance et différentes stratégies de globalisation de méthode d'ordre supérieur en fonction du temps de calcul en BigFloat

Donc les directions qui calculent $\nabla^3 f(x)uv$ prennent significativement plus de temps que l'algorithme de région de confiance classique pour atteindre un point stationnaire. Ce ralentissement est accentué si l'on prend n plus grand.

Donc malgré le fait que l'on observe une certaine tendance où les méthodes d'ordre supérieur permettent généralement d'épargner des itérations dans l'algorithme de région de confiance, les directions de Chebyshev et (Super)Halley ne sont pas encore calculées assez rapidement pour pouvoir offrir un réel avantage. La direction de Shamanskii est donc la seule direction d'ordre supérieur qui a été utilisée pour les sections suivantes.

Finalement, malgré le fait que les résultats dans les figures 4.1 et 4.2 soient en `BigFloat`, les courbes auraient une allure similaire en double précision. Nous ne présentons donc pas de résultats en double précision pour les méthodes de Chebyshev et (Super)Halley. Nous avons obtenus des résultats similaires avec la stratégie ARC qui ne sont pas présentés.

4.4.2 Comparaison d'algorithmes en Float64

Pour travailler avec des problèmes de grande dimension et des matrices creuses la seule direction d'ordre supérieur que l'on se permet d'utiliser est celle de Shamanskii.

Pour résoudre (3.1) et (3.9), la librairie HSL, et plus précisément le module MA57, est utilisée [HSL07]. Il s'agit d'une décomposition LDL^T avec pivotage qui est optimisée pour les matrices creuses. Les résultats numériques avec l'approche Shamanskii-BFGS (section 2.5) n'ont pas été encourageants avec des décompositions de matrices creuses. Cette méthode n'est donc pas présentée dans cette section.

Une sélection de 150 problèmes de la collection `CUTEst` [GOT15] est utilisée pour les tests à grande échelle . La sélection a été faite pour retirer les problèmes avec contraintes et les problèmes qui nécessitent plus de 5 minutes à résoudre. Les problèmes choisis

sont tels que $n \in [2, 10^5]$. La précision recherchée est de 10^{-8} lorsque les nombres sont représentées en `Float64`, autrement dit on considère qu'un point stationnaire est atteint lorsque $\|\nabla f(x)\| \leqslant 10^{-8}$.

Région de confiance

Les différentes stratégies proposées dans ce chapitre qui introduisent l'ordre supérieur dans l'algorithme 5 sont comparées à la version traditionnelle de l'algorithme de région de confiance. Nous utilisons les profils de performances introduits à la section 3.2.2 pour comparer les différentes méthodes. Nous utilisons le nombre d'itérations et le temps de calcul comme mesures de performance.

Dans la figure 4.3 nous comparons différentes combinaisons de λ_F et η_N dans la stratégie (4.2) avec un algorithme de région de confiance traditionnel. Le nombre d'itérations est utilisé comme mesure de performance.

Il n'y a pas de stratégie de globalisation et de combinaison de paramètres λ_F et η_N qui est systématiquement plus rapide que l'algorithme de région de confiance TR. En effet, comme nous l'avons vu dans les exemples numériques des sections 4.2 et 4.3 certaines combinaisons sont plus ou moins efficace en fonction du problème choisi et du point de départ. Cependant, la combinaison de $\lambda_F = 1000$ et $\eta_N = 0.8$ dans la stratégie (4.2) est généralement plus rapide que la version traditionnelle. Cette combinaison a été trouvée après avoir échantillonné différentes combinaisons de λ_F et η_λ.

Les stratégies (4.2) avec $\lambda_F = 1000$ et $\eta_N = 0.8$ et l'algorithme de région de confiance sont comparées en termes d'itération dans la figure 4.4. Donc pour la majorité des problèmes, il est avantageux d'utiliser la stratégie (4.2) avec $\lambda_F = 1000$ et $\eta_N = 0.8$ selon les tests numériques que nous avons réalisés. D'autant plus que toutes ces variantes résolvent un même nombre de problèmes.

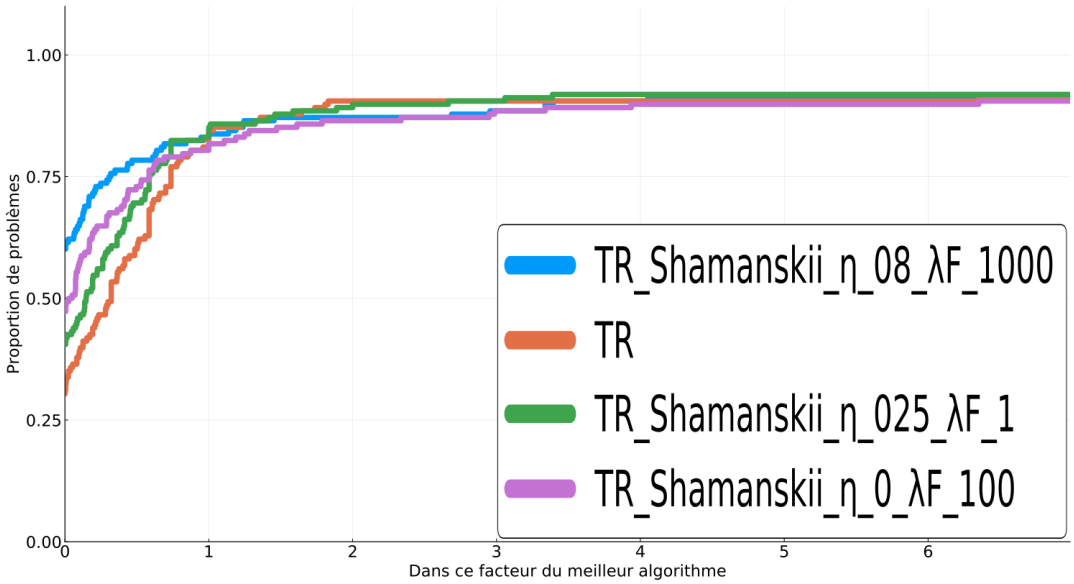

Figure 4.3 – Comparaison de l'algorithme de région de confiance et différentes stratégies de globalisation de méthode d'ordre supérieur en fonction du nombre d'itérations

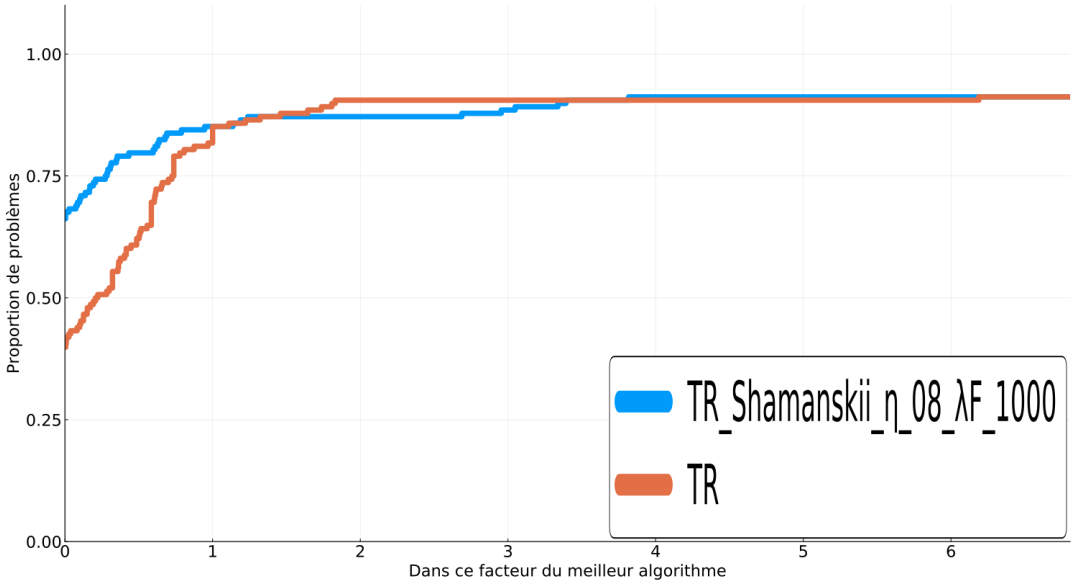

Figure 4.4 – Comparaison de l'algorithme de région de confiance et de la stratégie (4.2) avec $\lambda_F = 1000$ et $\eta_N = 0.8$ en fonction du nombre d'itérations.

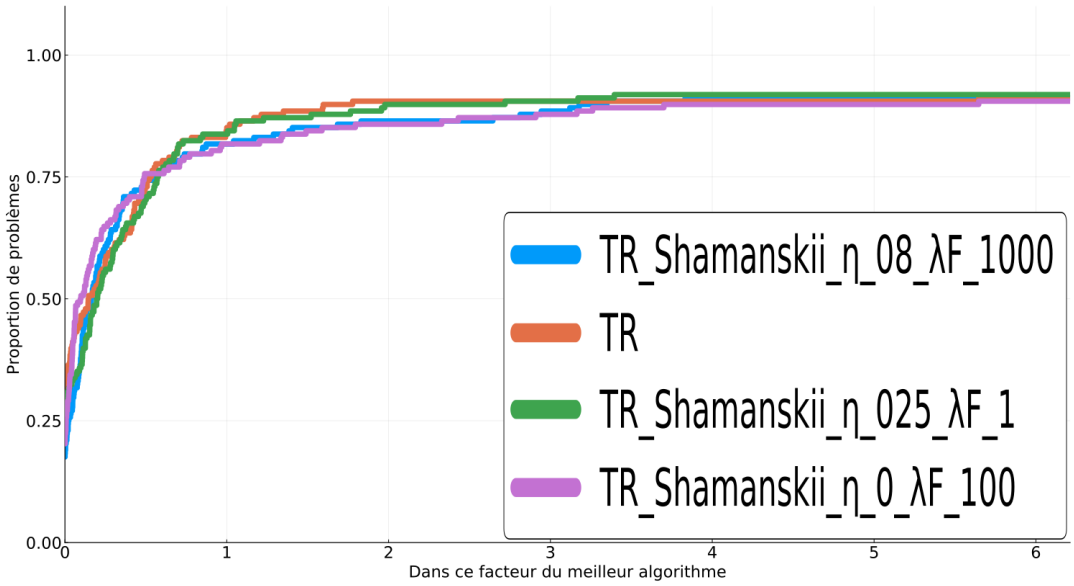

Figure 4.5 – Comparaison de l'algorithme de région de confiance et différentes stratégies de globalisation de méthodes d'ordre supérieur en fonction du temps de calcul.

La comparaison des mêmes stratégies de globalisation en termes de temps de calcul est consignée dans la figure 4.5. On remarque que selon cette métrique, toutes ces stratégies sont équivalentes. Donc, le gain d'itérations des nouvelles stratégies n'est pas au dépend du temps de calcul.

Donc, la stratégie (4.2) améliore généralement l'algorithme de région de confiance traditionnel en termes de nombre d'itérations pour un temps de calcul équivalent.

Algorithmes ARC

Les différentes stratégies proposées dans ce chapitre qui introduisent l'ordre supérieur dans l'algorithme 5 sont comparées à la version traditionnelle de l'algorithme ARC. Une analyse similaire aux algorithmes de région de confiance est présentée dans cette section.

La figure 4.6 présente une comparaison de l'algorithme ARC classique avec la stratégie

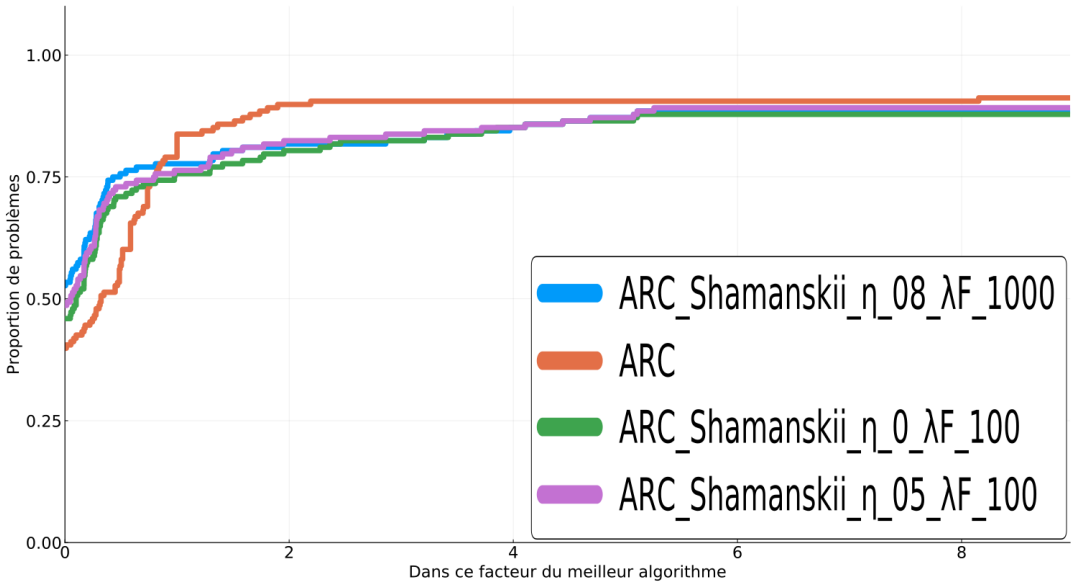

Figure 4.6 – Comparaison de l'algorithme ARC et différentes stratégies de globalisation de méthodes d'ordre supérieur en fonction du nombre d'itérations.

(4.6) utilisant différents λ_F et η_N.

Les différentes combinaisons de λ_F et η_N offrent toutes une accélération de l'algorithme ARC en termes d'itérations. Cette accélération est moins prononcée que pour les stratégies de globalisation des méthodes d'ordre supérieur par région de confiance. De plus on remarque que toutes les variantes résolvent environ 85% des problèmes. Elles sont donc équivalentes en termes de robustesse.

La comparaison avec le temps de calcul comme mesure de performance est présentée à la figure 4.7. Contrairement aux régions de confiance, les idées de globalisation ralentissent l'algorithme ARC traditionnel. Ce ralentissement peut s'expliquer par le fait que la direction d'ordre supérieur est calculée plus souvent qu'avec les algorithmes de région de confiance.

En général, la stratégie (4.6) est donc légèrement plus rapide en termes de nombre d'ité-

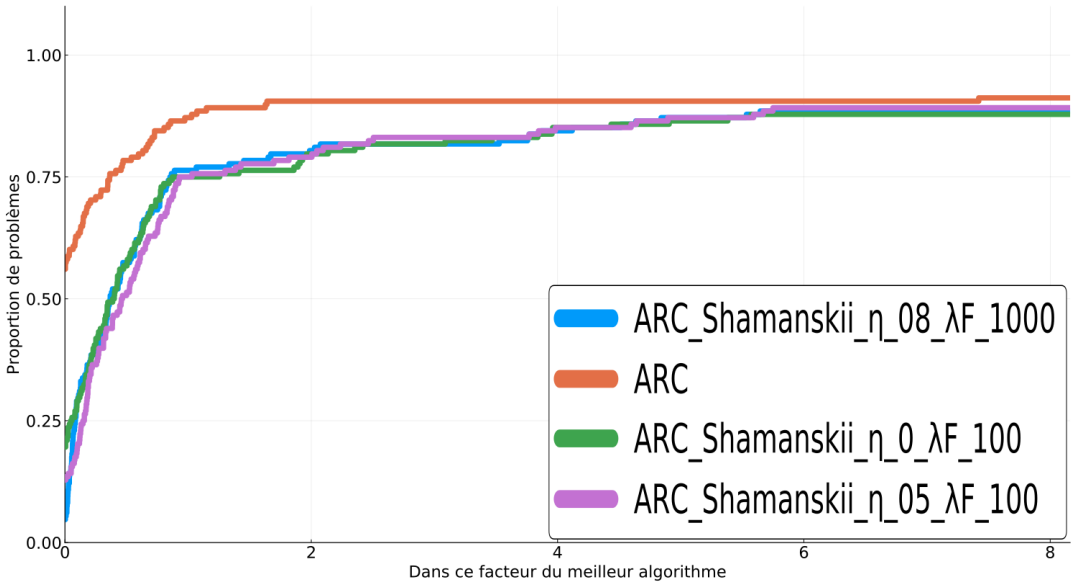

Figure 4.7 – Comparaison de l'algorithme ARC et différentes stratégies de globalisation de méthode d'ordre supérieur en fonction du temps de calcul.

rations et plus lente en termes de temps de calcul que l'algorithme ARC traditionnel.

CONCLUSION

L'objectif principal de ce mémoire était de revisiter les directions d'ordre supérieur et de les intégrer dans des stratégies de globalisation connues. Pour ce faire, nous avons présenté la méthode de Newton ainsi que les méthodes de Chebyshev, (Super)Halley et Shamanskii dans le contexte de la convergence globale. Nous avons présenté les difficultés liées à l'implémentation de ces méthodes ; entre autres, le calcul de l'information d'ordre trois pour les méthodes de Chebyshev et Halley. Deux pistes pour surmonter ces difficultés ont été introduites : les nombres hyperduaux et la différentiation automatique.

Les stratégies de recherche linéaire et de région de confiance, qui globalisent traditionnellement la direction de Newton, ont été présentées dans le troisième chapitre. Une attention particulière a été portée sur les méthodes de région de confiance. Entre autre, la résolution du sous-problème de région de confiance. Les stratégies ARC ont également été présentées, les similarités entre les stratégies de région de confiance et ARC ont été mises de l'avant. L'algorithme 5 présente une unification des stratégies ARC et région de confiance en un seul schéma.

Des nouvelles idées pour introduire des méthodes d'ordre supérieur dans des stratégies de région de confiance et ARC ont été introduites dans le quatrième chapitre. Ces stratégies consistent à analyser une direction d'ordre supérieur lorsque certains critères indiquent qu'elle pourrait accélérer l'algorithme de région de confiance standard. Ces stratégies res-

pectent les propriétés qui assurent la convergence théorique des stratégies traditionnelles. De nombreux exemples numériques exhibent les propriétés de convergence des nouvelles idées de globalisation. Finalement, des comparaisons sur des collections de problèmes ont été réalisées. Ces comparaisons indiquent qu'en termes de temps de calcul, introduire de l'ordre supérieur dans des schémas de globalisation connus ne modifie pas significativement la vitesse de convergence. Cependant, en termes d'itérations, introduire des directions d'ordre supérieur dans des stratégies de globalisation s'avèrent être généralement plus rapide que les algorithmes de globalisation traditionnels. Bien que ce gain d'itération est souvent asymptotique, l'impact des corrections d'ordre supérieur peut influencer la progression de l'algorithme bien avant d'être dans un voisinage d'une solution.

Différentes perspectives pour continuer ce projet de recherche sont possibles. Tout d'abord, il serait intéressant de pouvoir utiliser les directions de Chebyshev et Halley pour des fonctions où n est grand. Comme nous l'avons vu en petite dimension, ces directions peuvent être un meilleur choix que la direction de Shamanskii. Donc, mettre en place un outil de DA permettant de les utiliser lorsque n est grand est intéressant.

Une autre perspective intéressante serait d'introduire des directions d'ordre supérieur dans d'autres contextes. Par exemple, les directions d'ordre supérieur ont été introduites dans des algorithmes de points intérieurs. Il pourrait être intéressant d'introduire de l'ordre supérieur dans d'autres schémas algorithmiques.

Bibliographie

[Arm66] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1) :1–3, 1966.

[BEKS17] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia : A fresh approach to numerical computing. *SIAM Review*, 59(1) :65–98, 2017.

[BK77] J. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31 :163–163, 1977.

[Bro70] C. G. Broyden. The Convergence of a Class of Double-Rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1) :76–90, 1970.

[BS83] W. Baur and V. Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22 :317–330, 1983.

[CGT00] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, 2000.

[CGT11a] C. Cartis, N. I. M. Gould, and P. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. Part I : motivation, convergence and numerical results. *Mathematical Programming*, 127(2) :245–295, 2011.

[CGT11b] C. Cartis, N. I. M. Gould, and P. L. Toint. Adaptive cubic regularisation methods for unconstrained optimization. Part II : worst-case function- and derivative-evaluation complexity. *Mathematical Programming*, 130(2) :295–319, 2011.

[CH98] S. H. Cheng and N. J. Higham. A modified Cholesky algorithm based on a symmetric indefinite factorization. *SIAM Journal on Matrix Analysis and Applications*, 19(4) :1097–1110, 1998.

[CM90a] V. Candela and A. Marquina. Recurrence relations for rational cubic methods I : The Halley method. *Computing*, 44(2) :169–184, 1990.

[CM90b] V. Candela and A. Marquina. Recurrence relations for rational cubic methods II : The Chebyshev method. *Computing*, 45(4) :355–367, 1990.

[CMS99] P. L. Chebyshev, A. A. Markov, and N. Sonin. *Œuvres de P.L. Chebyshev*. Number vol. 1 in Œuvres de P.L. Chebyshev. Commissionaires de l'Académie impériale des sciences, 1899.

[DJM74] J. E. Dennis Jr. and J. J. More. Quasi-Newton methods, motivation and theory. Technical report, Ithaca, NY, USA, 1974.

[DJM82] J. E. Dennis Jr and E. Marwil. Direct secant updates of matrix factorizations. *Mathematics of Computation*, 38(158) :459–474, 1982.

[DM02] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2) :201–213, 2002.

[Dus08] J.-P. Dussault. La différentiation automatique et son utilisation en optimisation. *RAIRO - Operations Research - Recherche Opérationnelle*, 42(2) :141–155, 2008.

[Dus15] J.-P. Dussault. *Optimisation mathématique avec applications en imagerie*. Sherbrooke, Québec, Canada, 2015.

[Dus18] J.-P. Dussault. ARCq : a new adaptive regularization by cubics. *Optimization Methods and Software*, 33(2) :322–335, 2018.

[Dus19] J.-P. Dussault. A unified efficient implementation of trust-region type algorithms for unconstrained optimization. *INFOR : Information Systems and Operational Research*, pages 1–20, 2019.

[FA11] J. A. Fike and J. J. Alonso. The development of hyper-dual numbers for exact second-derivative calculations. In *AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting*, 2011.

[Fle70] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3) :317–322, 1970.

[Fle87] R. Fletcher. *Practical methods of Optimization*. John Wiley & Sons, New York, NY, USA, second edition, 1987.

[Gol70] D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109) :23–26, 1970.

[GOT15] N. I. M. Gould, D. Orban, and P. L. Toint. CUTEst : a Constrained and Unconstrained Testing Environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60(3) :545–557, 2015.

[GS16] N. I. M. Gould and J. Scott. A note on performance profiles for benchmarking software. *ACM Transactions on Mathematical Software*, 43(2) :1–15, 2016.

[GW08] A. Griewank and A. Walther. *Evaluating Derivatives : Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008.

[Hal94] E. Halley. *A new, exact and easy method of finding the roots of equations generally and that without any previous reduction*, volume 18. Philos. Trans. Roy. Soc. London, 1694.

[HSL07] HSL. *The HSL Mathematical Software Library*. STFC Rutherford Appleton Laboratory, 2007. `http://www.hsl.rl.ac.uk`.

[KD14] B. Kchouk and J.-P. Dussault. A new family of high-order directions for unconstrained optimization inspired by Chebyshev and Shamanskii methods. *Optimization Methods and Software*, 29(1) :68–87, 2014.

[LD15] M. Lubin and I. Dunning. Computing in operations research using Julia. *INFORMS Journal on Computing*, 27(2) :238–248, 2015.

[LS01] F. Lampariello and M. Sciandrone. Global convergence technique for the Newton method with periodic hessian evaluation. *Journal of Optimization Theory and Applications*, 111(2) :341–358, 2001.

[MGH81] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1) :17–41, 1981.

[MSA03] J. R. R. A. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3) :245–262, 2003.

[New69] I. Newton. *De analysi per aequationes numero terminorum infinitas*. Royal Society Library, 1669.

[OR00] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[Rap97] J. Raphson. *Analysis aequationum universalis seu ad aequationes algebraicas resolvendas methodus generalis, & expedita, ex nova infinitarum serierum methodo, deducta ac demonstrata*. typis Tho. Braddyll, prostant venales apud Johannem Taylor, ad insigne Navis in Coemeterio D. Pauli, 1697.

[Sha70] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111) :647–656, 1970.

[SS40] F. Simonart and T. Simpson. *Essays on Several Curious and Useful Subjects, in Speculative and Mix'd Mathematics : Illustrated by a Variety of Examples*. J. Nourse, 1740.

[Tra64] J. F. Traub. *Iterative methods for the solution of equations*. Prentice-Hall series in automatic computation. Prentice-Hall, 1964.

[Š67] V. E. Šamanskiĭ. On a modification of the Newton method. *Ukrains'kyi Matematychnyi Zhurnal*, 19(1) :133–138, 1967.

[Wol69] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2) :226–235, 1969.